# Entwicklung einer vereinfachten Flugphysik für modulare Flugzeuge

## Versuch der Erstellung eines realistisch wirkenden Prototyps

## Bachelor-Thesis

**zur Erlangung des akademischen Grades B.Sc.**


## Stephan Köhler

**2052966**

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik


Erstprüfer: Prof. Dr. Edmund Weitz

Zweitprüfer: Prof. Dr. Torsten Edeler

Hamburg, 11. 06. 2017

# Inhaltsverzeichnis

# Abstract

This work discusses the thesis, whether it is possible to create realistic flight physics by using simple technics only. For this a comparison between the prototype developed especially for this purpose will be compared with the necessary requirements for a realistic simulation of a flight. The goal is to successfully implement a realistic seeming flight characteristics based on simple calculations. Physically basics will be covered and their implementation or approach in the prototype will be described as well. Further effects and calculations, which are needed to achieve a realistic simulation, will be discussed.
The Airplane is constructed with a modular structure and its flight characteristics are generated from its components and their values. Before the start of the flight, the user can customize the airplane by choosing the components individually.
The Prototype is developed using the Unity Engine. However, none of the Unity implemented physically functions will be used. The programming language is C-Sharp.



# Zusammenfassung

In dieser Arbeit wird die Frage behandelt, ob es möglich ist, mit einfachen Mitteln eine realistisch wirkende Flugphysik zu erstellen. Der zu diesem Zweck erstellte Prototyp wird mit den Anforderungen an eine Simulation verglichen. Das Ziel ist es, ein möglichst realistisch wirkendes Flugverhalten mit einfachen Berechnungen zu erreichen. Physikalische Grundlagen werden erörtert und anschließend ihre Umsetzung oder Annäherung im Programm beschrieben. Des Weiteren werden weiterführende Effekte und Berechnungen erörtert, die für eine realistische Simulation bedacht werden müssen.
Das Flugzeug ist modular aufgebaut und sein Flugverhalten ergibt sich aus den verschiedenen Komponenten und ihren Werten. Vor dem Flug kann das Flugzeug vom Nutzer aus den einzelnen Komponenten zusammengesetzt werden.
Umgesetzt wird die Flugphysik mit der Unity Engine. Es werden keine in der Engine bereits implementierten Physikberechnungen verwendet. Als Programmiersprache wird C-Sharp genutzt.

# 1 Einleitung

Die Motivation zu dieser Arbeit besteht zum Einen aus dem generellen Interesse an Luftfahrt und der Idee sich selbst einen vereinfachten Flugsimulator zu schreiben, zum Anderen an dem Plan die Fähigkeiten im Bereich des Programmierens zu verbessern und somit die bisherigen gewählten Schwerpunkt des Studiums zu erweitern. Die anfängliche Idee einen größeren Fokus auf die Erstellung der Modelle zu legen wurde zu Gunsten des Simulationsvergleichs verworfen.
Die Frage die in dieser Arbeit beantworten werden soll lautet somit:
**Ist es möglich, als einzelne Person ohne fundierte physikalische Vorkenntnisse und mit geringer Programmiererfahrung innerhalb einer Bachelorarbeit einen recht realistisch wirkendes Flugverhalten darzustellen?**
Das Ziel ist es sämtliche physikalischen Funktionen selbst zu schreiben und nicht auf die in Unity bereits implementierte Physikengine zurück zu greifen. Auf diese Weise soll die volle Kontrolle über die Berechnungen behalten und die Berechnungen nicht an eine nicht einsehbare Blackbox abgeben werden.

Diese Arbeit beginnt mit der Erörterung der „Startbedingungen“ (Kapitel: 1.1), also den persönlichen Erfahrungen und Voraussetzungen, die als Grundlage dienen. Im folgenden Kapitel wird die genutzte „Unity Engine“ (Kapitel: 2) beschrieben. Dies erfolgt zuerst durch eine Erörterung der „Oberfläche“ (Kapitel: 2.1) und somit der Funktionen der Engine, danach folgt eine kurze Beschreibung des „Scriptsystems“ (Kapitel: 2.2).

Im dritten Kapitel wird auf den eigentlichen Prototypen des Flugsimulators eingegangen. Nach der Darstellung des „Funktionsumfangs“ (Kapitel: 3.1) des Programms erfolgt eine ausführlichere Beschreibung der „Entwicklungsschritte“ (Kapitel: 3.2), welche zu den einzelnen Teilbereichen weiterleitet. Es folgt eine Erörterung des generellen „Programmaufbaus“ (Kapitel: 3.3). Bevor die eigentlichen Berechnungen beschrieben werden können, muss das genutzte Koordinatensystem festgelegt werden, was in der Sektion „Einheiten und Koordinatensystem“ (Kapitel: 3.4) geschieht. In der darauf folgenden Sektion „Flugzeugkomponenten“ (Kapitel: 3.5) werden die einzelnen Komponenten beschrieben, aus denen sich das Flugzeug zusammensetzen lässt. Eine Erweiterung hierzu stellt die Sektion „Tragflächensegmente“ (Kapitel: 3.6) dar. Hier wird das erste Mal eine Grundlage zur korrekten Simulation beschrieben, welche jedoch lediglich in stark vereinfachter Form genutzt wird. Generell werden in den folgenden Sektionen häufig physikalische Grundlagen und Gesetzmäßigkeiten beschrieben, deren Umsetzung oder Annäherung im Programm danach benannt werden. Al-

ternativ handelt es sich dabei um mögliche Erweiterungen, welche später zusammengefasst werden. Das Kapitel Stabilisierung durch „Tragflächenausrichtung“ (Kapitel: 3.7) beschäftigt sich mit der durch die Bauform des Flugzeugs erzeugten Stabilisierung der Flugbahn. Im folgenden Kapitel „Flugzeugsteuerung“ (Kapitel: 3.8) werden die im Programm genutzten Möglichkeiten zur Steuerung des Flugzeugs benannt. Die nächste Sektion beschäftigt sich mit „dynamischem Auftrieb“ (Kapitel: 3.9), gefolgt von der Implementierung verschiedener „Anströmrichtungen“ (Kapitel: 3.10). Die Sektion „$C_W$-Wert“ (Kapitel: 3.11) beschreibt den formabhängigen Widerstand, was in „Luftwiderstand und Staudruck“ (Kapitel: 3.12) fortgeführt wird. Danach wird in der Sektion Luftdichte (Kapitel: 3.13) den Einfluss des Luftdrucks beschrieben. In der nächsten Sektion wird das Thema „Schwerkraft“ (Kapitel: 3.14)behandelt, danach folgt eine Beschreibung von Schub und der davon abhängigen „Beschleunigung“ (Kapitel: 3.15). In der folgenden Sektion geht es um „Schwerpunkt und Auftriebspunkt“ (Kapitel: 3.16), ein Kapitel welches eher zu möglichen Erweiterungen gehört. Die Erstellung der Umgebung wird in der nächsten Sektion „Landschaft und Wolken“ (Kapitel: 3.17) beschrieben. Als Abschluss des dritten Kapitels wird auf die größten angefallenen „Schwierigkeiten und Probleme“ (Kapitel: 3.18)eingegangen, die im Verlauf der Arbeit entstanden sind, gefolgt von einer Erörterung weiterer Schritte in der Sektion „Mögliche Erweiterungen“ (Kapitel: 3.19).

In dem vierten Kapitel werden die Alternativen und Bestehendes betrachtet. Dies erfolgt in der Sektion „Geschichte der Flugsimulatoren“ (Kapitel: 4.1), in welcher die wichtigsten Meilensteine in der Entwicklung von Flugsimulatoren benannt werden. Im Bereich Baukästen und Arcade Spiele werden die Spiele „Simple Planes“ (Kapitel: 4.2.1), „Kerbal Space Program“ (Kapitel: 4.2.2) sowie „Besige“ (Kapitel: 4.2.3) kurz vorgestellt. Danach folgt eine kurze Vorstellung der Flugsimulatoren „X Plane“ (Kapitel: 4.3.1) sowie des Microsoft „Flight Simulators“ (Kapitel: 4.3.2).
Abschließend folgt im fünften Kapitel das „Fazit“ der Arbeit (Kapitel: 5).

## 1.1 Startbedingungen

Das Programm entstand mit verhältnismäßig geringem fachspezifischem Vorwissen. Der bisherige gewählte Schwerpunkt im Studium lag im Gamedesign und Gestaltungsbereich. Zusätzliche Programmiererfahrung zu den Pflichtvorlesungen fehlte, somit ist die Umsetzung im Programm im Zusammenhang mit der Zielsetzung (einer realistisch wirkenden Flugsimulation ohne große Vorkenntnisse) zu sehen. Des Weiteren fehlte jegliche Erfahrung in der Modellierung dynamischer Systeme.
Als bestehende Grundlage für die Flugberechnungen diente eine zu zwei Dritteln abgeschlossene Segelflugausbildung, welche jedoch bereits 8 Jahre vor der Erstellung dieser Arbeit aus Zeitmangel abgebrochen werden musste. Ein generelles Interesse an dem Thema Luftfahrt war jedoch seit dem vorhanden und auch privat werden Flugsimulationen genutzt.
Die zur Erstellung des Programms genutzte Unityengine [Tec17b] diente bereits bei vorherigen Studienprojekten als Grundlage, weshalb sie relativ vertraut war.
Der Umgang mit Cinema 4D (Studentenversion: [Max17]), welches zur Erstellung der rudimentären Flugzeugkomponenten genutzt wurde, war ebenfalls durch vorherige Erfahrungen bei privaten Projekten vertraut.

# 2 Unity

Die Unity Engine, folgend Unity genannt, ist ein Entwicklungsumgebung, welche hauptsächlich zur Entwicklung von Spielen genutzt wird. Andere Anwendungen, zum Beispiel im Bereich Virtual Reality, sind jedoch ebenfalls möglich. Zum Zeitpunkt der Erstellung dieser Arbeit ist Version 5.5.0f3 aktuell[Tec17b], welche auch genutzt wird.

## 2.1 Oberfläche und Funktionen

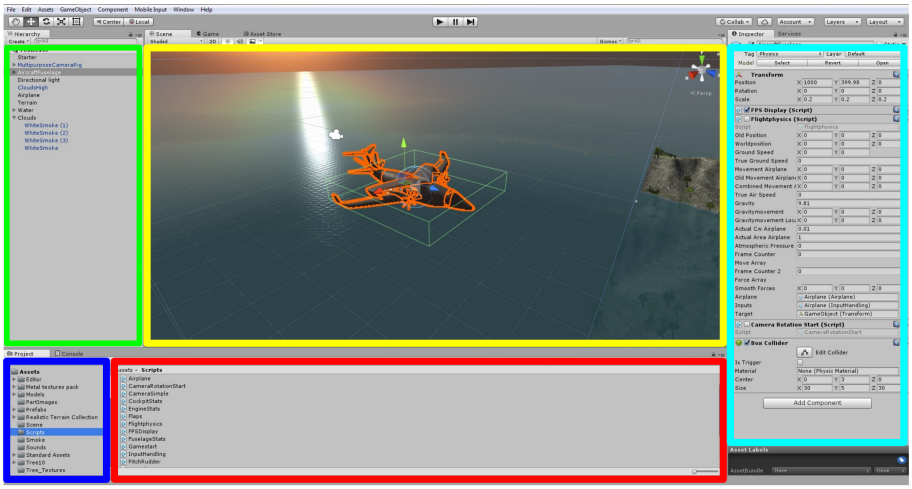


**Abbildung 2.1:** Die Oberfläche des Editors der Unity Engine [Koe17]

Die in Abbildung 2.1 dargestellte Oberfläche des Editors ist an den Aufbau gängiger 3D-Entwicklung- und Grafikprogramme angelehnt. Sie lässt sich nach den eigenen Bedürfnissen anpassen. Zentral befindet sich ein Fenster (gelb), in dem die aktuelle Szene dargestellt wird. Dieses geschieht wahlweise durch eine Editionsansicht der Szene (Reiter „Scene“), in der die Kamera frei bewegt und sämtliche Objekte manipuliert werden können, oder in der Spielansicht (Reiter „Game“). Beide Reiter befinden sich an der oberen linken Ecke der Szenenansicht. Hier wird die Sicht der in jeder Szene vorhandenen Standardkamera angezeigt, alternative Kameraobjekte können erstellt werden, dann erfolgt dort die Anzeige des Sichtfelds der zur Zeit aktiven Kamera. Wird das Spiel über den oberhalb der Szenenansicht gelegenen Playbutton gestartet,

wird es entweder im Szenenfenster abgespielt oder bei Aktivierung der Vollbildoption wird eine bildschirmfüllende Version angezeigt.
Im linken Bereich befindet sich die Hierarchie (grün markiert), in welcher sämtliche in der Szene befindlichen Objekte angezeigt werden. Dies betrifft statische Objekte, welche die Szene darstellen ebenso wie Lichtquellen, Spieleravatare oder die bereits erwähnten Kameraobjekte. Auch Abhängigkeiten der Objekte untereinander werden dargestellt, so werden im Programm zum Beispiel die einzelnen Flugzeugkomponenten(welche eigene Gameobjekte darstellen) einem Player Objekt untergeordnet, welches die errechneten Bewegungen ausführt. Auf diese Weise werden alle untergeordneten Gameobjekte gleichseitig angesprochen.
Auf der rechten Seite befindet sich die Detailansicht der Objekte (türkis markiert). Sie nennt sich Inspector und zeigt das jeweils in der Hierarchie ausgewählte Gameobjekt im Detail an. Hier werden alle Komponenten des Objekts angezeigt. Dies ist standardmäßig eine Transform Komponente, in welcher die aktuelle Skalierung, Position und Rotation des Objekts gespeichert wird. Diese Komponente muss zur Bewegung durch ein Script angesprochen werden. Scripte sind ebenfalls Komponenten und wirken jeweils auf das Gameobjekt, dem sie untergeordnet sind. Eine Manipulation anderer Objekte muss separat im Script beschrieben werden. Lichtquellen sind ein Beispiel für Objekte mit speziellen Komponenten, die eigene Parameter besitzen. Sie haben eine Lichtkomponente, in der dann einzelne Parameter per Script angesprochen und manipuliert werden können. Im Editor lassen sich die Werte ebenfalls einstellen, eine Manipulation zur Laufzeit ist darüber jedoch nicht möglich. Auch die speziellen Eigenschaften von Scripten sind sichtbar, so werden die Werte von im Script genutzten Variablen direkt angezeigt, wird das Spiel nicht im Vollbildmodus ausgegeben sind sogar die Veränderungen in Echtzeit zu sehen, was bei der Fehlersuche hilfreich sein kann. Auch Verknüpfungen von Scripten untereinander sind so sichtbar. So erhält zum Beispiel das „Airplane“- Skript beim Start der Simulation die ausgewählten Flugzeugkomponenten.
Im unteren linken Bereich befindet sich die Struktur der Projektordner (blau markiert). Hier lassen sich die Dateien organisieren, so bietet sich das Erstellen eines Ordners mit den gesamten Scripten an, ebenso wie eine Aufteilung der 3D-Objekte und Modelle der Szene sowie der Komponenten des Flugzeugs in verschiedene Ordner. Die Kombination auch von mehreren 3D-Modellen mit ihren Komponenten lässt sich in einem sogenannten „Prefab“ Prefab speichern. Dieses kann eine starke Arbeitserleichterung sein, so kann zum Beispiel ein Flugzeugmodell als vorher erstelltes Prefab in die Szene gezogen werden, ohne dass das Bewegungsscript oder die Lichtquellen die als Scheinwerfer genutzt werden, erneut zugewiesen werden müssen. Jedes Gameobjekt kann als Prefab abgespeichert werden. Dieser Bereich kann auch durch den Reiter „Console“ zu einer Anzeige für die Konsole umfunktioniert werden (rot markiert). Hier werden dann Fehlermeldungen ausgegeben. Auch in Skripten erstellte Debuginformationen können hier ausgegeben werden, ebenso wie die Werte von Variablen, was ebenfalls zur Fehlersuche oder Überprüfung einzelner Funktionen sinnvoll sein kann.

## 2.2 Scripte

Scripte lassen sich in Unity über die Eigenlösung Monodevelop erstellen und bearbeiten, es wird jedoch auch der Zugang zu einer aktuellen Version von Visual Studio geboten, welches in dieser Arbeit als Entwicklungsumgebung genutzt wird. Als Programmiersprachen stehen Unity Script (ähnlich Javascript), C-Sharp sowie Boo zur Verfügung. Im Programm wird C-Sharp genutzt. Jedes Script entspricht einer Klasse und kann auf die Komponenten der Gameobjekte zugreifen und diese manipulieren.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class NewBehaviourScript : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

**Abbildung 2.2:** Ein neu erstelltes C-Sharp Script [Koe17]

In Abbildung 2.2 ist ein neu erstelltes C-Sharp Script zu sehen. Die Klasse trägt den Defaultnamen „NewBehaviourScript“, welcher bei der Erstellung des Scripts geändert werden kann und dabei dann im Script direkt übernommen wird. Die neu erstellte Klasse erbt von „Monobehaviour“, welches die Basisklasse ist, die eine große Anzahl von Funktionen zur Verfügung stellt. Dieses Verhalten lässt sich deaktivieren. Zwei bereits in dem neuen Script vorhandene Funktionen sind „Start“ und „Update“. Start wird ein Mal beim ersten Durchlauf des Scripts angesprochen, hier sollte die Initialisierung stattfinden. Update hingegen wird bei jedem Durchlauf, also jedem Frame aufgerufen. Hier wird im Programm die eigentlichen Funktionen aufgerufen.

Weitere Möglichkeiten zur Steuerung des Programmablaufs sind zum Beispiel „Awake“, was aufgerufen wird, wenn die Instanz des Scripts geladen wird, oder „Late Update“, eine Funktion, die nach allen Update Funktionen aufgerufen wird. Dies ist nützlich, um den Ablauf des Scripts zu organisieren, so sollte eine Kamerabewegung in Late Update ausgeführt werden, nachdem die mögliche Positionsänderung des Objekts, dem die Kamera folgt, in Update berechnet wurde. Monobehaviour bietet noch eine Vielzahl weiterer Funktionen. Mit „OnCollisionEnter“ können zum Beispiel Collider überwacht werden, „OnMouseDown“ wird aufgerufen, wenn der Nutzer über einem GUI Element oder Button die Maustaste drückt.
Nach dem „using“ Befehl werden Namespaces importiert, die eine Vielzahl von Klassen beherbergen und so der Organisation Bibliothek dienen. „System.Collections“ beherbergt beispielsweise die „.Net“ Klassen und ermöglicht eine Organisation der Daten unter Anderem in einer Array List. Die Unity Scripting Reference: [Tec17a].

# 3 Das Programm und die Grundlagen

## 3.1 Funktionsumfang

Das Programm bietet dem Nutzer die Möglichkeit, sich ein Flugzeug, bestehend aus einer Auswahl verschiedener Komponenten, zusammen zu stellen. Das Flugzeug besteht aus fünf Komponenten (Cockpit, Rumpf, Flügel, Triebwerke sowie Heckpartie), von denen jeweils zwei verschiedene Versionen mit unterschiedlichen Werten zur Auswahl stehen.(Siehe Kapitel: 3.5) Der Nutzer hat hier Beispielweise die Wahl zwischen Propeller und Düsenantrieb. Außerdem haben alle Komponenten ein unterschiedliches Gewicht und unterscheiden sich in ihren $C_W$-Werten.


**Abbildung 3.1:** Der Auswahlbildschirm der Komponenten des Flugzeugs [Koe17]

Nach der Auswahl aller Komponenten mittels Anklicken der Vorschaubilder wird die Auswahl durch Betätigen der Leertaste bestätigt und der Flug beginnt. Der Startbildschirm mit der Komponentenauswahl ist in Abbildung 3.1 zu sehen.
Die Steuerung des Flugzeugs erfolgt durch die Pfeiltasten „Hoch“ und „Runter“ für das Höhenruder, „Rechts“ und „Links“ für das Querruder und „A“ und „D“ für das Seitenruder. Mit „W“ und „S“ wird der Schub geregelt, „Q“ und „E“ fahren die Landeklappen Stufenweise ein beziehungsweise aus. (Siehe Kapitel: 3.8)

Da das Programm über keine Kollisionserkennung verfügt, wird aus der Luft in freiem Fall gestartet. Abbildung 3.2 zeigt das Flugzeug im Flug, die Flugbahn ist hier bereits stabilisiert. In der oberen linken Ecke wird die aktuelle FpS Anzahl angezeigt, also die Anzahl berechneter Bilder in der Sekunde. Dieser Wert ist allerdings eher in der Entwicklung des Programms zur Erkennung von rechenintensiven Neuerungen interessant, als dass er dem Spieler, abseits der Erkenntnis ob der genutzte Rechner leistungsstark genug ist, etwas nützt.


**Abbildung 3.2:** DasFlugzeug im Flug [Koe17]

Im unteren linken Bereich befinden sich einige Anzeigen, die die aktuellen Werte des Flugzeugs wiedergeben. Diese sind die der aktuelle „Throttle“ Wert, also die Angabe wie viel Gas der Spieler zur Zeit gibt. der Wertebereich liegt hier zwischen 0 und 1. In der Abbildung 3.2 ruft der Spieler also ungefähr 80% der möglichen Motorleistung von 125 (keine reelle Einheit) ab.
Das alternative Turbinentriebwerk erzeugt eine maximale Leistung von 250. „CurrentEnginePower“ stellt diese aktuelle Leistung dar. Abhängig von der aktuellen Geschwindigkeit, die als „TrueAirSpeed“ ausgegeben wird, werden der tatsächlich erzeugte Schub (Ausgabe als „Thrust“) sowie der aktuell anliegende Luftwiderstand (Ausgabe als „CurrentDrag“) berechnet und ausgegeben. Mit „Hight“ wird noch die Flughöhe über dem Meeresspiegel angegeben. Die Angaben sind als Debuginformationen bei der Entwicklung sehr wichtig, einem Spieler würden Geschwindigkeit, Höhe und maximal noch der Throttle Wert reichen.

## 3.2 Entwicklungsschritte

Die Entwicklung begann mit der grundlegenden Flugphysik. Ein einfacher Würfel stellte zunächst das Flugzeug dar, auf welchen die Kraftvektoren wirken sollten. Die erste Idee war, die Kräfte in einzelnen Funktionen zu errechnen und jeweils direkt auf den Würfel anzuwenden. Nachdem ich mir das Beispielprojekt eines Flugzeugs in Unity angesehen hatte, entschied ich mich jedoch, den dort genutzten Ansatz zu übernehmen. Dieser sieht vor, dass alle Kräfte und Effekte zwar in einzelnen Funktionen errechnet werden, jedoch danach in einem einzigen Bewegungsvektor zusammengefasst werden, der erst danach auf das Flugzeug angewendet wird. Diese Vorgehensweise erschien mir weniger rechenintensiv, da das Flugzeug in jedem Durchlauf des Scrips nur eine zusammengefasste Bewegung ausführen muss, an Stelle vieler, für jede einzelne auf das Flugzeug wirkende Kraft.

Die ersten Funktionen, für die in entgegengesetzter Richtung wirkenden Kräfte, berechneten den Schub und den von der Geschwindigkeit abhängigen Widerstand. Ebenfalls abhängig von der Geschwindigkeit folgte danach die Funktion für den Auftrieb. Zu diesem Zeitpunkt verliefen die Kraftvektoren noch entlang der Flugzeugachsen. Der Vortrieb folgte der X-Achse in positiver Richtung, der Widerstand in negativer Richtung und der Auftrieb verlief entlang der positiven Y-Achse. Die Erkenntnis, dass sowohl Auftrieb als auch Luftwiderstand nicht von der Ausrichtung des Flugzeugs, sondern von der Anströmrichtung der Luft abhängen, erfolgte später, mehr dazu im weiteren Verlauf, zu diesem Zeitpunkt war noch keine Recherche in der Fachliteratur erfolgt. Der nächste Schritt war die grundlegende Implementierung von Schwerkraft, die Besonderheit hier ist die Ausrichtung am Weltkoordinatensystem, sie folgt daher im Gegensatz zu den anderen Kräften nicht der Rotation des Flugzeugs.
Als Nächstes habe ich die drei Rotationen implementiert, diese sind Rollen: Drehung um die Längsachse (Z-Achse), Nicken: Drehung um die Querachse (X-Achse) sowie Gieren: Drehung um die Hochachse (Y-Achse). Sie sind vom jeweiligen Ruderausschlag und der Geschwindigkeit direkt abhängige Rotationen. (Siehe Kapitel Flugzeugsteuerung: 3.8)
Nach dieser sehr vereinfachten Bewegungsimplementierung erfolgte der Aufbau der Szene. Zur besseren Einschätzbarkeit der ausgeführten Bewegung wurde eine Landschaft eingefügt. Den Würfel, welcher bisher das Flugzeug dargestellt hat, habe ich durch ein unsichtbares Gameobjekt ersetzt, dem die erste Iteration der Flugzeugkomponenten untergeordnet wurde. Zu diesem Zeitpunkt hatten sie jedoch noch keine weitere Funktion als das Flugzeug optisch darzustellen.

Bisher befand sich der gesamte Programmcode in dem „Flightphysics“ -Script. Zur Vorbereitung für die Erweiterung des Programms habe ich dann die Bearbeitung der Steuereingaben des Nutzers in ein separates Script ausgelagert, ebenso wie die Werte des Flugzeugs, welche sich jetzt im „Airplane“ -Script befinden. Das war die Grundlage, um die gesamten Eigenschaften des Flugzeugs aus einzelnen Scripten für jedes

Bauteil bestimmen zu können, welche ihre Werte an das „Airplane“ -Script übergeben. (Siehe Kapitel Aufbau des Programms: 3.3)

Nachdem die Aufteilung abgeschlossen war, folgte eine Analyse bezüglich des sich noch sehr unrealistisch anfühlenden Flugverhaltens. Nach einiger Recherche ([BH13], [BAL11][Ost12] sowie [Kas03]) stellten sich zwei Punkte als besonders wichtig heraus. Der erste Punkt sind aerodynamische Effekte, die dafür sorgen, dass sich das Flugzeug (wenn die konstruktionsbedingten Voraussetzungen erfüllt sind) in seine Bewegungsrichtung ausrichtet.(Siehe Kapitel Aufbau und Ausrichtung der Tragflächen: 3.7) Da die genaue Berechnung der Effekte für diese Arbeit deutlich zu aufwändig ist, habe ich eine von der Fluggeschwindigkeit abhängige generelle Drehung in Bewegungsrichtung implementiert. Diese ist stark vereinfacht, der resultierende Effekt ist jedoch erstaunlich realitätsnahe.
Die zweite wichtige Änderung betrifft die bereits erwähnte Ausrichtung von Auftrieb und Luftwiderstand. (Siehe Kapitel Anströmrichtung:3.10 sowie Auftrieb: 3.9, CW-Wert: 3.11 und Luftwiderstand:3.12).Vor der Anpassung an die Anströmrichtung war es dem Flugzeug bei einem Flug mit hohem Anstellwinkel fast möglich auf der Stelle zu schweben, da Auftrieb Schub und Schwerkraft sich gegenseitig im Dreieck ausgleichen konnten. Hier wird eine weitere Schwierigkeit sichtbar, welche bei der gesamten Entwicklung auftrat: Die Stärke der einzelnen Kräfte wie Schwerkraft, Schub und Luftwiderstand müssen nach jeder Änderung eines Wertes neu aufeinander abgestimmt werden, um realistisch wirkende Ergebnisse zu erzielen.(Siehe Kapitel Schwierigkeiten und Probleme: 3.18

Nachfolgend habe ich eine erste Version der Animationen erstellt. Diese bestehen aus Scripten, welche die Einzelnen Objekte rotieren. Als Grundlage dient hier die Steuerungsvariable der jeweiligen Achse. Somit folgen die Höhen-, Seiten- und Querruder optisch den Steuerbefehlen des Nutzers. Die Bewegungen unterscheiden sich in den jeweils möglichen Maximalausschlägen, abhängig von dem jeweiligen Ruder. Die Flaps (Landeklappen) reagieren ebenfalls auf die Steuereingaben, sie haben vier verschiedene Stufen, zwischen denen flüssig gewechselt werden kann. Auch hier dient die Statusvariable als Grundlage, weshalb die Animation die genaue aktuelle Wirkung widerspiegelt, nicht zwingend die aktuelle Nutzereingabe. So dauert es etwas, bis die Flaps sich von einem Status zum nächsten bewegt haben.
In Abbildung 3.3 ist das Flugzeug im Flug zu sehen, während die Landeklappen voll ausgefahren sind und der Nutzer vollen Querruderausschlag gibt, das Flugzeug rollt also mit maximaler Geschwindigkeit um die Längsachse.
Der nächste Schritt war die Implementierung der Komponentenauswahl. Der Nutzer kann mit der Maus eine der beiden Komponentenvarianten auswählen. Seine Auswahl wird sofort an dem dargestellten Flugzeug angezeigt. Dies erfolgt durch Aktivieren der ausgewählten Komponente, sowie Deaktivieren der anderen. (Abbildung 3.1) Bei Bestätigung der Auswahl durch den Nutzer aktiviert das „Gamestart“ -Script die beim Start des Programms noch deaktivierten „Flightphysics“ und „Airplane“ -

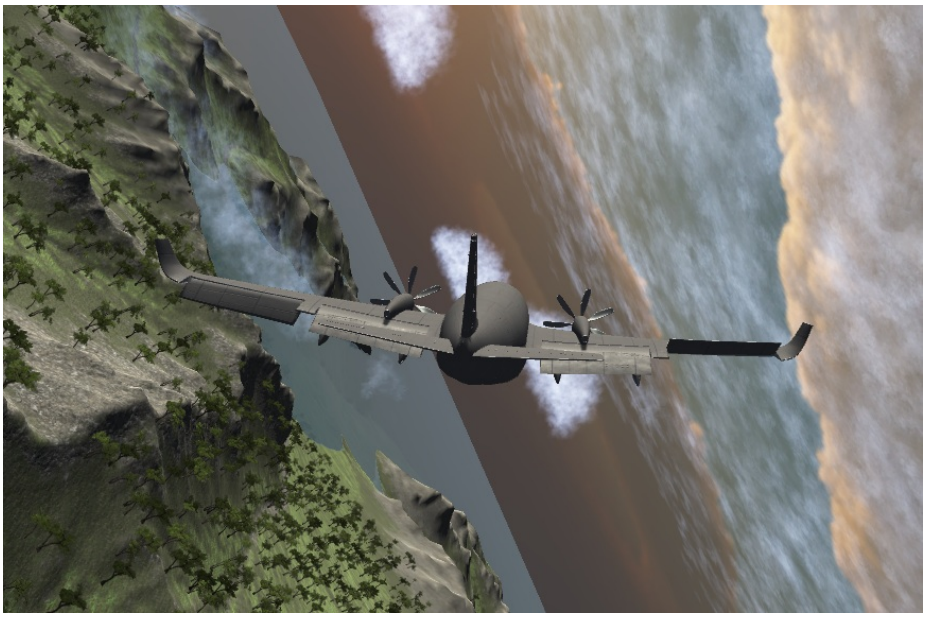


**Abbildung 3.3:** Voll ausgefahrene Flaps sowie voller Querruderausschlag [Koe17]

Scripte.(Siehe Kapitel Aufbau: 3.3)
Der nächste Schritt war Recherche für diese Arbeit. Als hauptsächliche Quellen dienen die Bücher „Flugregelung“ in der dritten Auflage von Rudolf Brockhaus, Wolfgang Alles und Robert Luckner, sowie „Physics for Game Developers“ in zweiter Auflage von David M. Bourg und Bryan Bywalec ([BH13], [BAL11]). Im Verlauf der Recherchen erhielten bisher intuitiv erstellte Lösungen eine fundierte Grundlage, was zu einer leichten Überarbeitung der bisherigen Berechnungen führte. Diese entsprachen allerdings zum Großteil bereits der Buchvorlage, lediglich anders benannt oder formatiert. Danach folgte eine Implementierung einfacher Soundeffekte. Genutzt werden drei unter der Creative Commons 0 Lizenz auf `https://www.freesound.org/` bereitgestelle Sounds. Hierbei handelt es sich um die Motorsounds für die beiden Triebwerksarten, sowie ein allgemeines Windgeräusch. Die Triebwerks Sounds passen sich in Lautstärke und pitch Wert an die aktuellen Triebwerksleistungseingaben des Nutzers an (throttle). Das Windgeräusch verhält sich auf Basis der aktuellen Geschwindigkeit identisch. Es folgte eine Überarbeitung der Schwerkraftberechnung, die zuvor sehr viel komplizierter aufgebaut war, aber trotzdem nicht wie gewünscht funktioniert hat.
Der letzte Schritt war eine leichte Aufbereitung der Optik des Programms. In diesem Zuge habe ich die Größe des Terrains angepasst sowie es mit einer Wasseroberfläche umgeben, welche den Effekt einer Insel erzeugt. Das Wasser wurde hierbei aus den Beispielprojekten die der Unity Engine beiliegen entnommen. Danach erfolgte die Erstellung von volumetrisch wirkenden Wolken sowie einem Nebeleffekt. (Siehe Kapitel Landschaft: 3.17)

## 3.3 Aufbau

Im Nachfolgenden wird der grundsätzliche Aufbau des Programms sowie der Zusammengang der einzelnen Scripte untereinander beschrieben. Veranschaulicht wird dies durch die Grafik 3.4. Die farbigen Pfeile markieren jeweils folgende Zusammenhänge:

- Schwarzer Pfeil
  Ein Script stellt seine feste eingetragenen oder errechneten Werte einem anderen Script zur Verfügung.
- Roter Pfeil
  Ist das Ausgangsscript aktiv wird ein Userinterface erzeugt.
- Grüner Pfeil
  Das Ausgangsscript aktiviert ein anderes Script.
- Blauer Pfeil
  Das Ausgangsscript wählt welche Variante der Werte im Scripts aktiviert wird und somit einem anderen Script zur Verfügung steht.

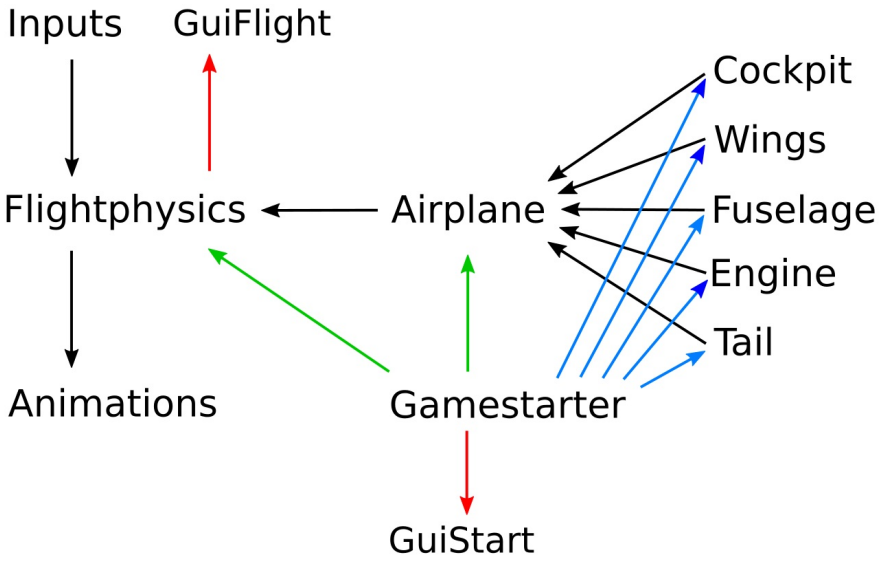



**Abbildung 3.4:** Der schematische Aufbau des Programms [Koe17]

Der Kern des Programms ist das Script „Flightphysics“. In diesem werden die Berechnungen zur Physik und daraus resultierend die Lage und Bewegung des Flugzeugs berechnet. Berechnet werden hier Schub, $C_W$-Wert und daraus resultierend der Luftwiderstand, Auftrieb sowie Schwerkraft und eine Vereinfachung mehrerer aerodynamischer Effekte, die dafür sorgen, dass sich das Flugzeug in Bewegungsrichtung ausrichtet. Auch die Auswirkungen der Nutzereingaben zur Steuerung werden hier verarbeitet.

Als Grundlage zur Berechnung dienen die Flugzeugeigenschaften, welche in dem „Airplane“ -Script zusammengefasst werden. Hier wird das Gesamtgewicht ermittelt, sowie die beiden zusammengefassten $C_W$-Werte, die Fläche aber auch der Schub der Triebwerke sowie die Multiplikatoren für die Wirksamkeit der Steuerflächen gesammelt.
Die Teilwerte stammen aus den einzelnen Komponenten -Scripts (Cockpit, Fuselage, Engine, Wings und Tail) Hier sind die Werte der verschiedenen Versionen der Komponenten gespeichert.
Die Komponentenauswahl erfolgt durch den Nutzer in dem Script „Gamestarter“. Dadurch wird festgelegt, welcher Wertesatz genutzt und welches Modell geladen wird. Außerdem aktiviert das Script nach abgeschlossener Konfiguration des Flugzeugs die vorher deaktivierten Scripte „Flightphysics“ und „Airplane“.
Nutzereingaben zur Steuerung des Flugzeugs werden in dem Script „Inputs“ verarbeitet. Die daraus resultierenden Steuerbefehle werden dann an „Flighphysics“ übergeben.

Das Script „Gamestarter“ erzeugt ein GUI, welches dem Nutzer die Zusammenstellung des Flugzeugs sowie den Spielstart erklärt. Außerdem zeigt es die zur Wahl stehenden Flugzeugkomponenten an. Dieses wird nach erfolgtem Start durch das von „Flighphysics“ erzeugte GUI ersetzt. Hier sind in Textform Fluginformationen zu sehen wie Schub, Geschwindigkeit und Flughöhe.
Die Animationen am Flugzeug, wie sich bewegende Ruder oder die ausfahrbaren Flaps, basieren auf dem „Flightphysics“ -Script und setzen so direkt die Steuerbefehle des Nutzers um.

## 3.4 Einheiten und Koordinatensystem

Bevor auf einzelne Berechnungen eingegangen werden kann, muss geklärt werden, in welchen Einheiten gerechnet werden soll, sowie welches Koordinatensystem genutzt wird.
Die Unityengine nutzt ein linkshändisches Koordinatensystem mit den Achsenbezeichnungen X (Rot), Y (Grün) und Z(Blau). Die folgenden Beispiele beziehen sich auf die Blickrichtung beziehungsweise Ausrichtung des Flugzeugs. Die X-Achse verläuft waagerecht entlang der Flügel, rechts ist ihr Wertebereich positiv, links negativ. Die Y-Achse steht senkrecht, oben ist ihr Wertebereich positiv, unten negativ. Die Z-Achse verläuft der Länge nach durch das Flugzeug, nach vorne ist ihr Wertebereich positiv, nach hinten negativ.
Wie umfangreich und komplex die benötigten Berechnungen für eine möglichst realistische Betrachtung des Flugverhaltens werden, wird bei der Definition des Koordinatensystems im Buch Flugregelung deutlich ([BAL11] Kapitel 1.2.1: Schreibweisen). Genau festgelegt werden die Definitionsregeln in den Din Normen 9300 sowie der ISO 1151 Part 1 bis 9. Hier wird ein orthogonales rechtshändiges Koordinatensystem mit den Bezeichnungen x,y und z genutzt. Ein Kraftvektor besteht aus den Komponenten X,Y und Z. Der Geschwindigkeitsvektor V hat die Komponenten u,v und w. Die Komponenten der Momentvektoren verlaufen positiv in Richtung der Achsen und haben die Bezeichnungen L,M und N. Die gesamte Drehung des Flugzeugs wird mit p, q und r beschrieben. Die nachfolgende Grafik 3.5 zeigt im oberen Bereich das in Unity genutzte linkshändische Koordinatensystem samt genutzter Farbkodierung, sowie darunter die an eine Zeichnung im Buch Flugregelung angelehnte Visualisierung des eben beschriebenen rechtshändigen Koordinatensystems.

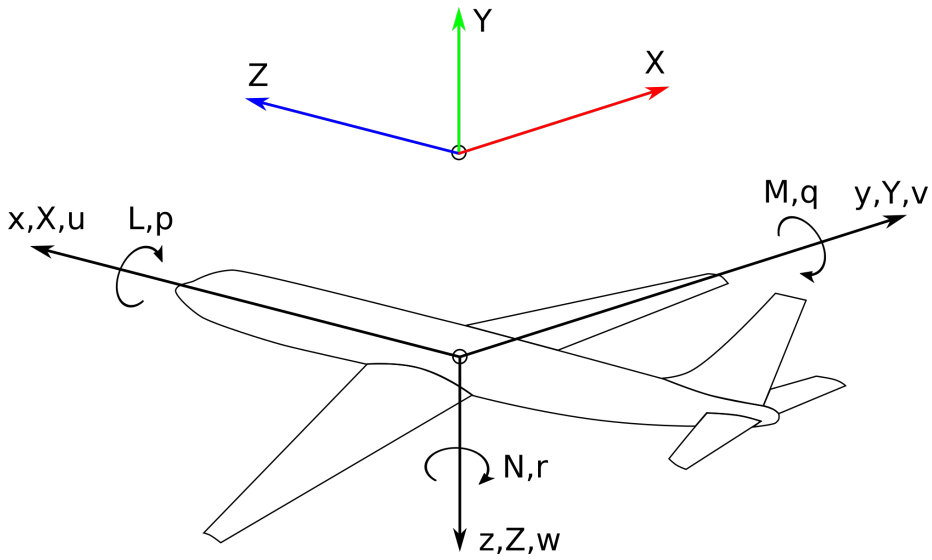



**Abbildung 3.5:** Das Koordinatensystem in Anlehnung an eine Zeichnung im Buch Flugregelung [BAL11]

## 3.5 Flugzeugkomponenten

Das Programm empfängt den Nutzer mit dem Auswahlbildschirm (Abbildung 3.1) der Flugzeugkomponenten. Hier stehen jeweils zwei Versionen der benötigten Komponenten zur Auswahl. Nachfolgend die Aufteilung des Flugzeugs in die Komponenten Cockpit (Cockpit), Rumpf (Fuselage), Tragflächen (Wings), Heckpartie(Tail) sowie Triebwerk (Engine).

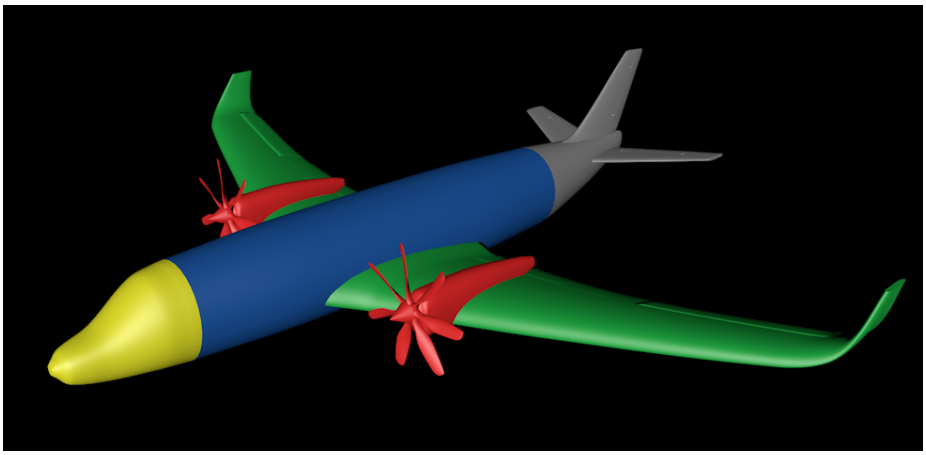


**Abbildung 3.6:** Die Komponenten des Flugzeugs [Koe17]

Abbildung 3.6 zeigt die einzelnen Komponenten farblich markiert in Cinema 4D. Nachfolgend eine Erörterung ihrer Aufgaben und Eigenschaften:

- Cockpit (Gelb)
  Vom Cockpit aus wird das Flugzeug gesteuert. Als besondere Eigenschaft hat es die Variable „throttlechangespeed“, welche die Reaktionsgeschwindigkeit der Triebwerke auf die Steuereingaben beschreibt. Die weiteren Eigenschaften sind ein frontaler und ein aufwärts gerichteter $C_W$-Wert (cwForwardCockpit, cwDownCockpit), eine frontale sowie eine aufwärts gerichtete Fläche (AreaForwardCockpit, areaDownCockpit) und das Gewicht (weightCockpit).
- Rumpf (Blau)
  Im Rumpf wird die Nutzlast des Flugzeugs untergebracht, er verfügt nur über die normalen Eigenschaften aller Komponenten, diese sind ein frontaler und ein aufwärts gerichteter $C_W$-Wert (cwForwardFuselage, cwDownFuselage), eine frontale sowie eine aufwärts gerichtete Fläche (AreaForwardFuselage, areaDownFuselage) und das Gewicht (weightFuselage).

- Tragflächen (Grün)
  Die Tragflächen sorgen für den Auftrieb, der bei ausreichender Geschwindigkeit durch Ausgleichen der Schwerkraft das Flugzeug in der Luft hält. Als besondere Eigenschaft haben sie daher die Variable „liftWings“, welche einen Auftriebsbeiwert beinhaltet. Da die Querruder ein Teil der Tragflächen sind, ist die Variable „rollEffektWings“ ebenfalls eine besondere Eigenschaft. Sie beschreibt wie wirkungsvoll die Steuerung über die Querruder ist. Die weiteren Eigenschaften sind ein frontaler und ein aufwärts gerichteter $C_W$-Wert (cwForwardWings, cwDownWings), eine frontale sowie eine aufwärts gerichtete Fläche (AreaForwardWings, areaDownWings) und das Gewicht (weightWings).
- Heckpartie (Grau)
  In der Komponente Heckpartie sind der hintere Teil des Flugzeugs und das Höhenruder sowie das Seitenruder zusammengefasst. Sie dient der Stabilisierung und Steuerung des Flugzeugs. Die besonderen Eigenschaften sind in den Variablen „yawEffect“ für das Seitenruder und „pitEffect„ für das Höhenruder gespeichert. Beide regeln wie bereits gesagt, wie wirkungsvoll die einzelnen Ruder sind. Die weiteren Eigenschaften sind wieder ein frontaler und ein aufwärts gerichteter $C_W$-Wert (cwForwardTail, cwDownTail), eine frontale sowie eine aufwärts gerichtete Fläche (AreaForwardTail, areaDownTail) und das Gewicht(weightTail).
- Triebwerke (Rot)
  Die Triebwerke liefern den Schub, welcher das Flugzeug antreibt. Ihre besondere Eigenschaft ist daher die Variable „maxPowerEngine“, welche den maximalen Schub beschreibt, den die Triebwerke erzeugen können. Die restlichen Eigenschaften sind wiederum ein frontaler und ein aufwärts gerichteter $C_W$-Wert (cwForwardEngine, cwDownEngine), eine frontale sowie eine aufwärts gerichtete Fläche (AreaForwardEngine, areaDownEngine) und das Gewicht(weightEngine).

## 3.6 Tragflächensegmente

In dem Programm werden die beiden Tragflächen als eine Einheit betrachtet. Alle Berechnungen erfolgen daher nur einmal gemeinsam für beide Tragflächen. Bei einer realistischen Simulation müssten die Eigenschaften der Tragflächen einzeln betrachtet werden. Außerdem sollten die einzelnen Tragflächen in kleinere Segmente unterteilt werden, da sich ihre Form (das Profil)und der Anstellwinkel und damit die Eigenschaften an verschiedenen Bereichen unterscheiden. Je feiner die Unterteilung erfolgt, desto genauer, aber auch aufwändiger, sind die Berechnungen.
So erhöht sich zum Beispiel die Schränkung (der Anstellwinkel) der Tragfläche zu den Tragflächenenden. Da Tragflächen nur bis zu einem bestimmten Anstellwinkel Auftrieb erzeugen, reißt so die Strömung beim Überschreiten des Winkels nicht schlag-

artig an der gesamten Tragfläche ab, sondern erfolgt nach und nach an den äußeren Enden der Tragflächen beginnend. Dies ermöglichte eine Korrektur des Piloten bei drohendem Strömungsabriss und erhöht somit die Sicherheit des Flugzeugs ([Kas03] Kapitel 9: Konstruktive Flughilfen).
Wie in dem Kapitel Auftrieb (3.9) beschrieben, erzeugt die Form der Tragflächen oberhalb einen Unterdruck sowie unterhalb Überdruck. Dieses Druckungleichgewicht möchte sich ausgleichen, was an den Tragflächenenden in Form von starken Wirbeln geschieht. (Teilweise sind diese sogar sichtbar und so stark, dass sie unter dem Begriff Wirbelschleppen nachfolgende Flugzeuge behindern) Um diesen Effekt abzumildern, sind Tragflächen zu den äußeren Enden neutraler geformt, sie erzeugen dort also weniger Auftrieb. Des Weiteren helfen sogenannte Winglets an den Flügelspitzen die Wirbelbildung zu verringern. Verwirbelungen erzeugen generell erhöhten Widerstand, weshalb durch die Winglets auch der Gesamtluftwiderstand verringert wird und die Effizienz steigt. In Abbildung 3.7 ist eine schematische Darstellung einer Tragfläche zu sehen, welche zu ihrem äußeren Ende eine starke Schränkung aufweist sowie ein neutrales Profil besitzt.

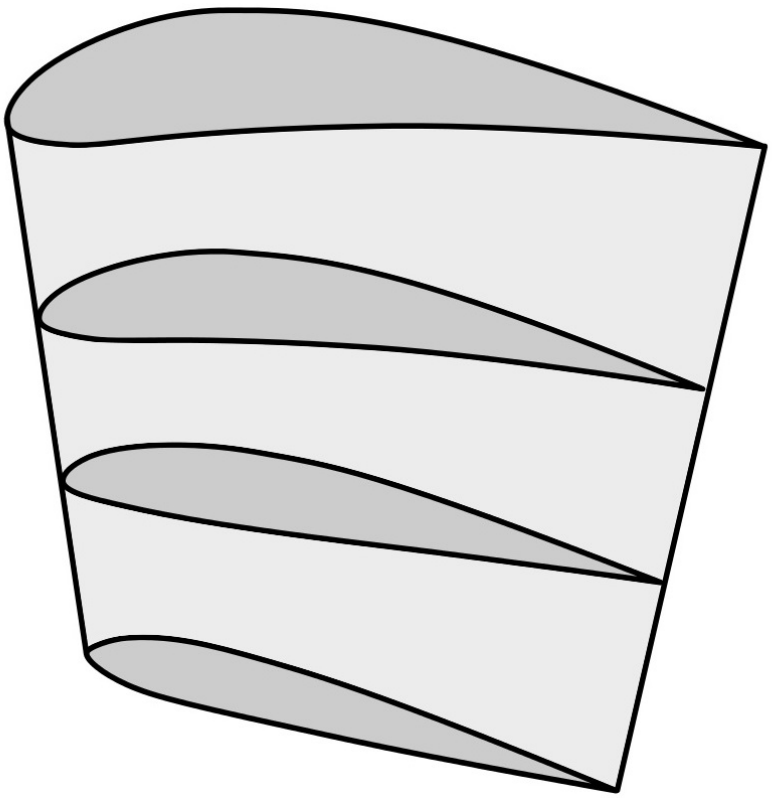


**Abbildung 3.7:** Eine Tragflächen mit zum äußeren Ende steigender Schränkung und neutraler Profilform [Koe17]

## 3.7 Stabilisierung durch Tragflächenausrichtung

Ein weiterer Grund für separate Betrachtung der einzelnen Tragflächen ist ihre Ausrichtung. Sie sind häufig in leichter V-Form angebracht, steigen also zu den Enden an. Das erzeugt eine automatische Stabilisierung und Ausrichtung des Flugzeugs in eine waagerechte Fluglage, da ein waagerecht liegende Tragfläche maximalen Auftrieb liefert ([Kas03] Kapitel 9: Konstruktive Flughilfen).

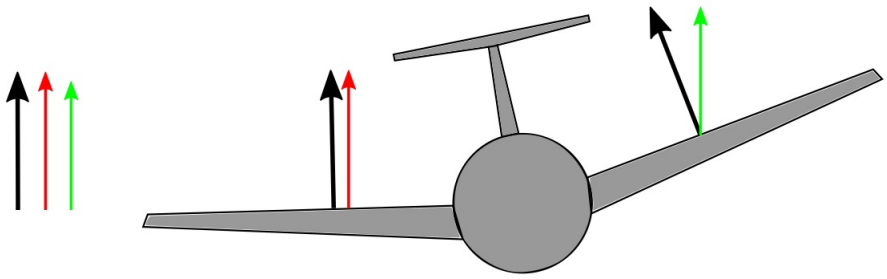


**Abbildung 3.8:** Tragflächen in V- Anordnung in leichter Schräglage [Koe17]

Sie erzeugt also im Vergleich zu der dann ansteigenden anderen Tragfläche mehr Auftrieb, was eine Rotation des Flugzeugs zur Folge hat, bis beide durch identische Winkel identischen Auftrieb liefern und sich die Momente ausgleichen. Somit wird das Flugzeug um die Z-Achse stabilisiert. Der schwarze Pfeil in Abbildung 3.8 stellt den gesamten Kraftvektor für den Auftrieb einer Tragfläche dar. Dieser liegt orthogonal zur Tragflächenrichtung an. Rot und grün sind jeweils die nach oben gerichteten Teilvektoren. Bei der waagerecht ausgerichteten Tragfläche entspricht der Teilvektor dem Gesamtvektor, bei der anderen ist er nur ein Teil und somit kürzer.

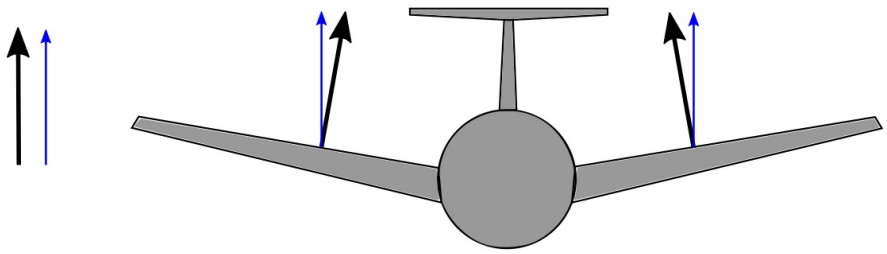


**Abbildung 3.9:** Tragflächen in V- Anordnung in stabilisierter Lage [Koe17]

In stabilisierter Fluglage sind beide blauen Teilvektoren gleich lang (Abbildung 3.9), jedoch minimal kürzer als der Gesamtvektor.

Einen ähnlichen Effekt erzeugt die Pfeilung der Tragflächen, also ihre Anwinkelung nach hinten. Sollte das Flugzeug schräg fliegen, in einem sogenannten schiebendem Flugzustand, so ist eine Tragfläche weiter nach vorne gedreht und wird somit frontal angeströmt, sie erzeugt so mehr Luftwiderstand als die weiter nach hinten gerichtete und schräg angeströmte zweite Tragfläche. Das sorgt auch hier für eine Ausrichtung und Stabilisierung, diesmal um die Y-Achse (Abbildung 3.10). Die frontal angeströmte Tragfläche wird durch ihren höheren Luftwiderstand stärker abgebremst als die andere Tragfläche.

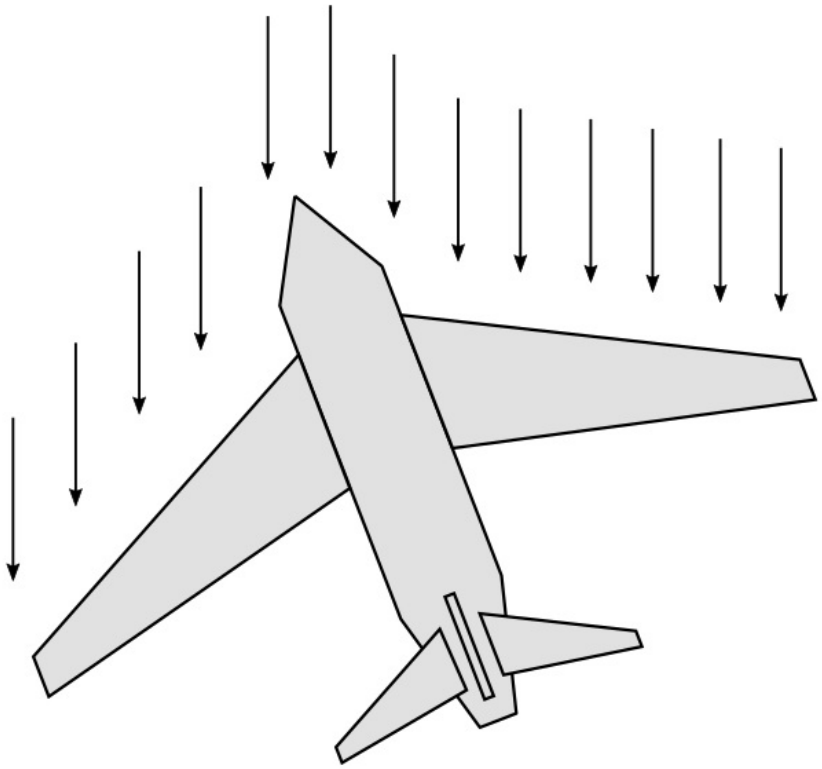


**Abbildung 3.10:** Flugzeug in schiebendem Flugzustand [Koe17]

Die Ausrichtung der Tragflächen sorgt also für eine Stabilisierung der Flugbahn des Flugzeugs. Sind die Winkel jedoch größer, steigt zwar der Stabilisierungseffekt, es geht allerdings auch Auftrieb verloren. Somit bestimmt der Einsatzzweck die Form der Flügel und ihre Ausrichtung. Eine starke Pfeilung der Tragflächen wird vor allem bei Militärflugzeugen genutzt, die auch in höherem Überschallbereich operieren können, damit aber auch durch den geringeren Auftrieb eine signifikant höhere Mindestfluggeschwindigkeit aufweisen als Beispielweise ein Segelflugzeug, welches schon bei sehr geringen Geschwindigkeiten genug Auftrieb erzeugt, dessen Höchstgeschwindigkeit allerdings im Vergleich stark limitiert ist.

Da in dem Programm keine Strömungsberechnung vorgenommen wird und auch die beiden Tragflächen lediglich als eine gemeinsame einzelne Einheit betrachtet werden, sind die zuvor beschriebenen Stabilisierungseffekte nicht korrekt umsetzbar. Allerdings entspricht ihre Wirkung grob einer generellen Ausrichtung des Flugzeugs in seine Bewegungsrichtung. Diese Rotation lässt sich jedoch darstellen.
Zur Umsetzung habe ich mir ein leeres Gameobjekt als Hilfe erstellt, welches dem Flugzeug untergeordnet ist und sich vor diesem befindet. Es bewegt sich also wie ein unsichtbarer Bestandteil des Flugzeugs mit und befindet sich auf der Z-Achse in positiver Richtung verschoben. In dem Codebeispiel 3.7 subtrahiere ich die Position des Hilfsobjekts von dem errechneten Bewegungsvektor des Flugzeugs in diesem Frame. Das Ergebnis wird normalisiert, der folgende Faktor bestimmt die Stärke der Rotation. Anschließend lasse ich das Flugzeug um die X- und Y- Komponente des neuen Vektors rotieren. Der Faktor „Time.deltaTime“ beschreibt die Zeit, die zur Berechnung des letzten Frames benötigt wurde, dadurch wird die Berechnung unabhängig von der Bildrate und das Ergebnis ist trotz unterschiedlicher Bildrate auf verschieden starken Systemen identisch.

```
     private void TurnToVelocity()
{
    turnToMovement = ((movementAirplane -
       target.transform.localPosition ).normalized)*60;
    transform.Rotate(-turnToMovement.y * Time.deltaTime,
       turnToMovement.x * Time.deltaTime, 0);
}
```

## 3.8 Flugzeugsteuerung

Ein Flugzeug wird unter Anderem durch verschiedene Klappen gesteuert, welche die Aerodynamik verändern und somit eine Änderung der bisherigen Bewegung auslösen. Die wichtigsten Steuerelemente, welche auch in dem Programm dargestellt werden, sind die Landeklappen (Flaps), das Höhenruder (Elevator), das Seitenruder (Yaw Rudder) sowie das Querruder (Aileron).

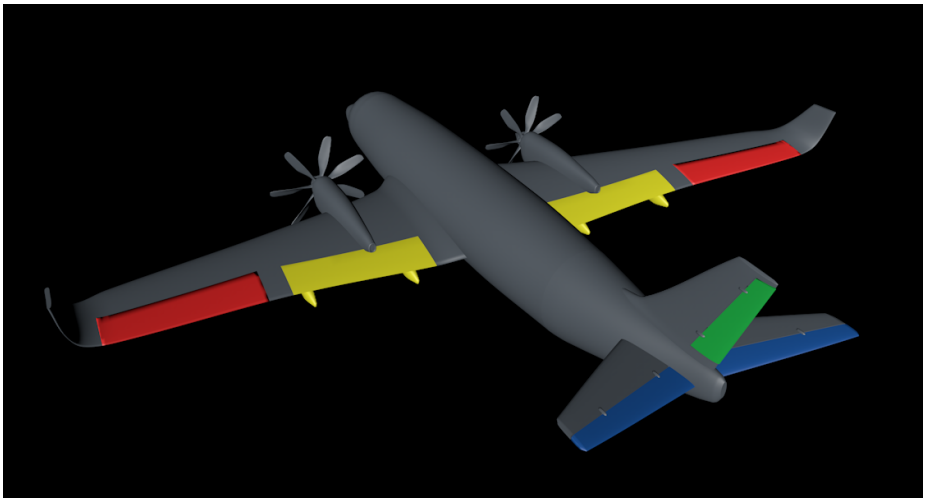


**Abbildung 3.11:** Die Steuerflächen des Flugzeugs [Koe17]

In Abbildung 3.11 sind die Steuerflächen farblich markiert, nachfolgend wird ihre Funktion beschrieben.

- Landeklappen (Gelb)
  Die Landeklappen sind Bereiche der Tragfläche, die bei Bedarf ausgefahren werden können. Hierbei verlängern sie die Tragfläche nach hinten und erweitern sie gleichzeitig nach unten. Somit ändert sich in dem Bereich das Flügelprofil und der Anstellwinkel, was den Auftrieb (Kapitel: 3.9), aber auch den Luftwiderstand (Kapitel: 3.12) erhöht. Im Programm haben die Landeklappen vier mögliche Positionen: Von 0 (komplett eingefahren, kein Effekt) bis 3 (verdoppelter Auftrieb und Luftwiederstand der Tragflächen)
- Höhenruder (Blau)
  Das Höhenruder befindet sich am hinteren Ende des Flugzeugs. Es steht meistens waagerecht und bewegt sich um die X-Achse, was eine Rotation des Flugzeugs um diese auslöst.
- Seitenruder (Grün)
  Das Seitenruder steht senkrecht am hinteren Bereich des Flugzeugs. Es rotiert um die Y-Achse und erzeugt damit eine Rotation des gesamten Flugzeugs um diese Achse.

- Querruder (Rot)
  Die Querruder befinden sich an den äußeren Tragflächenenden. Sie rotieren in entgegengesetzter Richtung um die X-Achse, erzeugen damit allerdings eine Rotation des gesamten Flugzeugs um die Z-Achse.

Alle Ruder verändern die Aerodynamik des Flugzeugs, was Momente an unterschiedlichen Achsen auslöst und so die Lagewinkel des Flugzeugs ändert ([BAL11] Kapitel 1.2.4: Flugzeugsteuerung). Jede Steuereingabe hat auch Auswirkungen auf den Widerstand, was in der Regel zu einer Erhöhung führt. Das Einleiten einer Kurve durch das Querruder erzeugt so zum Beispiel ein Moment um die Y-Achse entgegen der gewünschten Kurvenrichtung, obwohl die Querruder eigentlich eine Drehung um die Z-Achse auslösen. Dies nennt sich negatives Wendemoment und wird dadurch ausgelöst, dass der Flügel, der durch das Querruder mehr Auftrieb erhält, auch mehr Luftwiderstand erzeugt und damit stärker gebremst wird als der sich durch verringerten Auftrieb senkende Flügel, welcher auch weniger Luftwiderstand erzeugt ([Kas03] Kapitel: Technik des Fliegens: Negatives Wendemoment). Dieser Effekt müsste bei einer genauen Simulation beachtet werden (er würde sich automatisch ergeben), findet in dem Programm aber keine Beachtung, da nicht jede Klappe einzeln simuliert, sonder nur der gewünschte Steuereffekt auf das gesamte Flugzeug angewandt wird. Nachfolgend als Beispiel die Umsetzung beim Höhenruder: 3.8

```
private void ControlPitch()
 {
     pitchAngle = Mathf.Clamp((pitchAngle -
        inputs.getPitchInput() * -airplane.getPitchEffect() *
        Time.deltaTime), -1.35f, 0.95f);
     pitchAngle -= pitchAngle / 20;
 }
```

Die aus dem „inputs“ Script stammenden Steuereingaben des Spielers werden mit dem im „airplane“ Script gesammelten „pitchEffekt“ multipliziert. Anschließend folgt die übliche Multiplikation mit der Zeitvariablen. „Mathf.Clamp“ beschränkt den möglichen Wertebereich, in diesem Fall also die maximalen Ruderausschläge. Die -1,35 entsprechen dem Maximalwert beim Heben des Flugzeugbugs, also einer Kurve nach oben. Da diese Bewegung positive G-Kräfte auslöst, welche der menschliche Körper deutlich besser verträgt als negative G-Kräfte, ist der Wert höher als die 0.95 für das Senken. Anschließend wird der Wert um ein Zwanzigstel reduziert, was ohne weitere Steuereingaben dazu führt, dass sich das Ruder wieder in eine neutrale Position begibt, diese Bewegung beginnt stark und wird immer schwächer, was ein sanftes Flugverhalten ergibt.

Die Anwendung der Rotationen erfolgt wie folgt:3.8

```
transform.Rotate(pitchAngle * (movementAirplane.z + 50)
   * 0.35f * Time.deltaTime, yawAngle *
   (movementAirplane.z + 50) * 0.35f * Time.deltaTime,
   rollAngle * (movementAirplane.z + 50) * 0.4f *
   Time.deltaTime);
```

Die einzelnen errechneten Steuereingaben werden mit der Z-Komponente (nach vorne) des Bewegungsvektors verrechnet. Mit steigender Geschwindigkeit steigt somit auch die Effektivität der Ruder. Dies ist eine stark vereinfache Annäherung an eine sonst nötige Strömungsberechnung.

## 3.9 Dynamischer Auftrieb (Lift)

Als dynamischer (Aerodynamischer) Auftrieb wird die Kraft bezeichnet, die senkrecht zur Anströmungsrichtung auf einen entsprechend geformten Körper wirkt. Dieser Auftrieb kann auf zwei unterschiedlich Arten entstehen, in der Praxis wirkt bei einer Tragfläche eines Flugzeugs meistens eine Kombination beider Arten.
Die erste Art entsteht durch den Anstellwinkel der Tragfläche, also den Winkel, mit welchem sie von der Luft umströmt wird. Die Luft wird nach unten abgelenkt, die entstehende entgegengesetzt wirkende Kraft ist der Auftrieb. (3. Newtonsches Gesetz: Eine Kraft erzeugt eine gleich große in entgegengesetzter Richtung verlaufende Gegenkraft) Die abgelenkte Strömung ist in Abbildung 3.12 dargestellt.

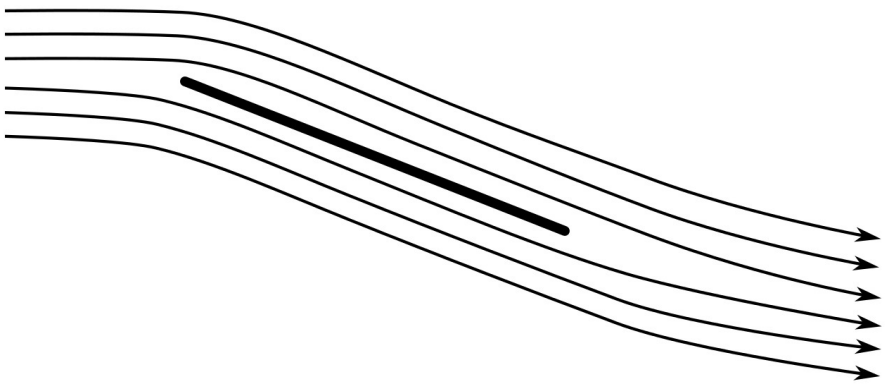


**Abbildung 3.12:** Die Umströmung eines neutral geformten Körpers bei hohem Anstellwinkel [Koe17]

Die zweite Variante Auftrieb zu erzeugen beruht auf der Form des umströmten Körpers. Abbildung 3.13 illustriert dies. Die anströmende Luft teilt sich am Staupunkt (grün). Der obere Teil der Luft (blau) legt eine größere Strecke zurück als der untere (rot). Dadurch wird die obere Luft beschleunigt, was zu einem Unterdruck führt (Gesetz von Bernoulli: Je größer die Strömungsgeschwindigkeit eines Gases ist, desto kleiner ist der statische Druck), während unterhalb ein leichter Überdruck entsteht. Die Kombination aus Überdruck unter der Tragfläche (ca 1/3) und dem Unterdruck oberhalb (ca 2/3) erzeugen den Auftrieb ([Kas03] Kapitel: Technik 3. Auftriebserzeugung).

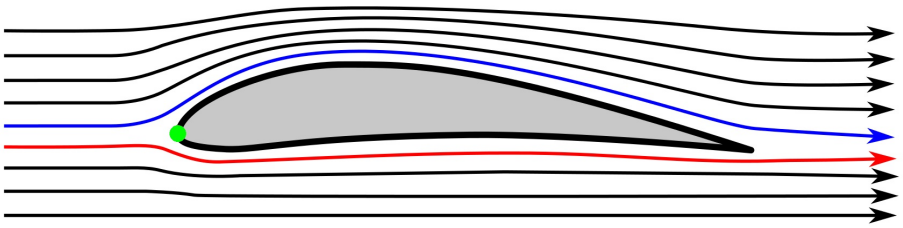


**Abbildung 3.13:** Die Umströmung einer Tragfläche bei niedrigem Anstellwinkel [Koe17]

Die Formel für den Auftrieb lautet: ([BAL11] Kapitel 1.2.2 Auftrieb und Widerstand)

$$A = \frac{p}{2} * V_A^2 * C_A * S \tag{3.1}$$

$A$ = Auftrieb
$S$ = Fläche der Tragfläche
$C_A$ = Auftriebsbeiwert
$p$ = Luftdichte
$V$ = Anströmgeschwindigkeit, also die aktuelle Geschwindigkeit des Flugzeugs gesehen zur Luft.

Die Umsetzung im Programm erfolgt wie folgt: 3.9. Die Faktoren hinter „trueAirspeed“ sowie „currentLift“ dienen lediglich zum Abstimmen der Größen untereinander, da ohne Einheiten gerechnet wird. Im If-Zweig wird der normale Fall betrachtet, dass sich das Flugzeug vorwärts bewegt (entlang der Z-Achse). Hier wird die Formel 3.1 angewendet. Im Else-Zweig wird der Fall abgefangen, dass sich das Flugzeug nicht vorwärts bewegt, in dem Fall wird die Z-Komponente des Gesamtbewegungsvektors auf Null gesetzt, um negative Werte zu vermeiden (Auf ihm basiert die in „trueAirspeed“ angegebene Geschwindigkeit)

```
private void CalculateLift()
{
    if ( movementAirplane.z > 0f)
    {
        currentLift = ((airplane.getM_Lift() * flapsFactor) *
           (atmosphericPressure/2)*((trueAirSpeed / 100 )*
           (trueAirSpeed / 100) ) );
        currentLift *= 100;
    }
    else
    {
        movementAirplane.z = 0f;
    }
}
```

Gleichzeitig entsteht ein sogenannter Strömungswiderstand, Genaueres im Kapitel (Widerstand 3.12).
Für einen stationären Geradeausflug muss der erzeugte Auftrieb $A$ die Gewichtskraft $G$ $(mg)$ ausgleichen. Um dies bei wechselnden Geschwindigkeiten zu erreichen, lässt sich der Auftriebsbeiwert der Tragfläche beeinflussen. Dazu hat ein Flugzeug je nach Typ unterschiedliche Möglichkeiten. In diesem Programm werden lediglich die Landeklappen (Flaps) betrachtet. Diese erhöhen sowohl den formbedingten Auftrieb, als auch den Anstellwinkel in der Flügelsektion. Somit gilt:

$$C_A = \frac{mg}{p/2 * V_A^2 * S} \tag{3.2}$$

Ebenso wie die Tragflächen, deren Auftriebsangriffspunkt vor dem Schwerpunkt liegt, erzeugt auch das Höhenleitwerk bei erhöhtem Anstellwinkel Auftrieb, der zum Gesamtauftrieb des Flugzeugs beiträgt. Die Form ist jedoch neutral, wodurch bei direkter Anströmung ohne Anstellwinkel (Siehe Kapitel Anströhmrichtung 3.10) kein Auftrieb erzeugt wird. Somit wandert der Angriffspunkt des Auftriebs bei Erhöhung des Anstellwinkels nach hinten. Dies sorgt wiederum durch den Momentausgleich für eine Verringerung des Anstellwinkels ([BAL11] Kapitel 1.2.3 Momentengleichgewicht). Ohne Beeinflussung durch das Höhenruder pendelt sich das Flugzeugs also in einem Zustand mit geringerem Anstellwinkel ein. Ändert sich der Anstellwinkel nicht mehr, spricht man von einem stationären Gleichgewicht. Die Steuerung per Höhenruder beeinflusst die Form und damit die Aerodynamik des Höhenleitwerks, was dann zu erhöhtem Auftrieb, aber bei entgegengesetzter Steuerung auch zu Abtrieb führen kann.
Die Formel für das stationäre Gleichgewicht lautet: 3.3.

$$A_0 = \frac{p}{2} * V_{A0}^2 * S * C_A * (\alpha_0) = G \tag{3.3}$$

$\alpha_0$ = Anstellwinkel des Höhenruders
Die Geschwindigkeit des Flugzeugs lässt sich also durch den Anstellwinkel beeinflus-

sen, der wiederum durch das Höhenruder beeinflusst werden kann. Im Programm wird dieser aerodynamische Effekt durch eine leichte Rotation des Flugzeugs in seine Bewegungsrichtung angenähert. Das ist stark vereinfacht, jedoch ist der Effekt sehr ähnlich. Die Umsetzung wird in Kapitel Tragflächenausrichtung3.7 beschrieben, der Programmcode ist in 3.7 aufgeführt.

## 3.10 Anströmrichtung

Von der Richtung, aus der das Flugzeug im Flug angeströmt wird, hängen direkt die Richtungen für Auftrieb und Luftwiderstand ab. Der Winkel zwischen der Ausrichtung in Längsachse (Z-Achse positiv) und der tatsächlichen Bewegungsrichtung (ohne Rotation um die Längsachse, also im Geradeausflug) nennt sich Anstellwinkel. Auftrieb und Schwerkraft müssen sich ausgleichen, wenn das Flugzeug sowohl seine Höhe als auch seine Geschwindigkeit beibehalten soll. Ein erhöhter Anstellwinkel steigert den Auftrieb bei identischer Geschwindigkeit, allerdings damit ebenfalls den Luftwiderstand. ([Kas03] 3.4 Der Einfluss des Anstellwinkels auf den Auftrieb)(Siehe Kapitel Auftrieb 3.9) Gut zu erkennen ist dies im Landeanflug von Flugzeugen, hier wird mit stark erhöhtem Anstellwinkel geflogen, damit die Fluggeschwindigkeit möglichst gering ausfallen kann. Der Bug des Flugzeugs zeigt also nach oben, obwohl das Flugzeug an Höhe verliert.
Im Programm wird der Anstellwinkel nicht errechnet, sondern ausgelesen. Die Grundlage ist der Bewegungsvektor, auf dessen Basis die Richtung des Auftriebs sowie des Widerstandsvektors bestimmt werden. Der Auftrieb liegt orthogonal zur Bewegungsrichtung an (nach oben). Der Luftwiderstand wirkt entgegengesetzt der Bewegungsrichtung.
Nachfolgend wird der Code für die Berechnung der Auftriebsrichtung durch Bildung des Kreuzprodukts der Bewegungsrichtung mit dem festen Vektor (1,0,0) gezeigt 3.10. Das Ergebnis wird normalisiert, um einen Richtungsvektor zu erhalten. Dieser wird danach mit dem Wert für den Auftrieb multipliziert.

```
var liftDirectional = new Vector3(0,0,0);
if (movementAirplane.z != 0)
{
     liftDirection = Vector3.Cross(movementAirplane,
     new Vector3(1, 0, 0)).normalized;
}
else
{
    liftDirection = new Vector3(0, 1, 0);
}
liftDirectional.x = currentLift * liftDirection.x;
liftDirectional.y = currentLift * liftDirection.y;
liftDirectional.z = currentLift * liftDirection.z;
```

## 3.11 $C_W$-Wert

Der $C_W$-Wert (Widerstandsbeiwert oder Anströmungswiderstandsbeiwert) ist ein Koeffizient, der den durch seine Form bestimmten Strömungswiderstand eines Gegenstandes beschreibt. Er bezieht sich lediglich auf die Form des Körpers, zur Errechnung des tatsächlichen aktuellen Widerstands werden noch die Fläche sowie die Anströmgeschwindigkeit benötigt ([Kas03] 4.3 Der Formwiderstand). Ein weiterer Faktor ist die Dichte des umströmenden Materials, in diesem Fall der Luft, also der Luftdruck.(Siehe Kapitel Luftdichte: 3.13)
Der $C_W$-Wert ist abhängig von der Geschwindigkeit. Er steigt bei wachsender Geschwindigkeit zunächst nur leicht an, bis sich die Geschwindigkeit der Schallgeschwindigkeit nähert, dort ist ein starker Anstieg des $C_W$-Werts zu beobachten. In diesem Programm wird der Einfachheit halber nur der Unterschallbereich betrachtet und der $C_W$-Wert somit als konstant angenommen.
Beispielhafte $C_W$-Werte:

- Fallschirm: 1,33
- Stehender Mensch: 0,78
- Tragfläche eines Flugzeugs: 0,08
- Stromlinienförmiger Tropfen: 0,02

Typischer Weise werden $C_W$-Werte im Windkanal ermittelt, sie lassen sich allerdings auch berechnen, was sehr aufwändig sein kann.
Im Programm bekommt jede Flugzeugkomponente zwei $C_W$-Werte, sowie zwei Flächenwerte zugeordnet. Der erste $C_W$-und Flächenwert beziehen sich auf eine frontale Anströmrichtung, wie in Abbildung 3.14 dargestellt. Dies entspricht einem horizontalen Flug, bei dem sich Auftrieb und Schwerkraft genau ausgleichen. So wird der geringste Widerstand erreicht, der Flug ist also so effizient wie möglich.

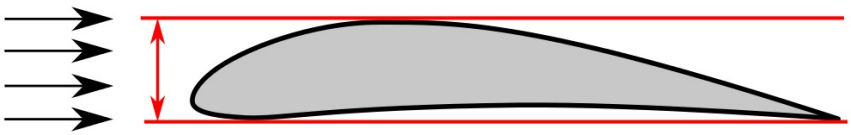


**Abbildung 3.14:** Anströmung einer Tragfläche mit einem Anstellwinkel von 0° [Koe17]

Der zweite Wert beschreibt jeweils eine um 90° gedrehte Anströmung direkt von unterhalb, dargestellt in Abbildung 3.15.
Im Flug wird das Flugzeug mit einem Anstellwinkel angeströmt, der je nach Flugzustand irgendwo zwischen den beiden Werten liegt. Der Einfachheit halber wird im Programm davon ausgegangen, dass sich der $C_W$-Wert linear zwischen den beiden

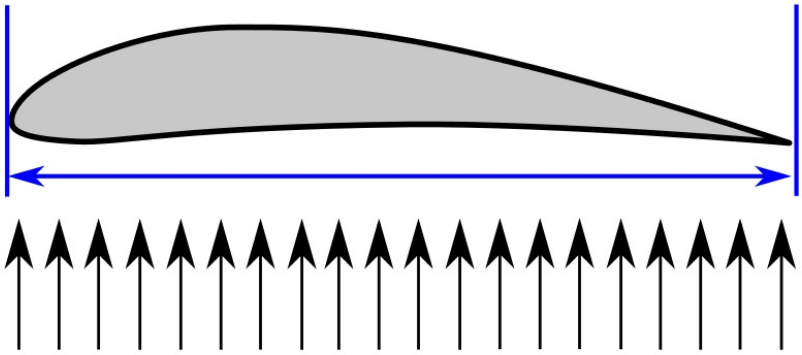


**Abbildung 3.15:** Anströhmung einer Tragfläche mit einem Anstellwinkel von 90° [Koe17]

Extremen ändert, dies ist jedoch in der Realität nicht der Fall. Die nachfolgende Grafik 3.16 zeigt einen sehr hohen Anstellwinkel von 30°, der im regulären Flugbetrieb nicht erreicht wird, im Kunstflug sind allerdings auch deutlich höhere Anstellwinkel möglich.

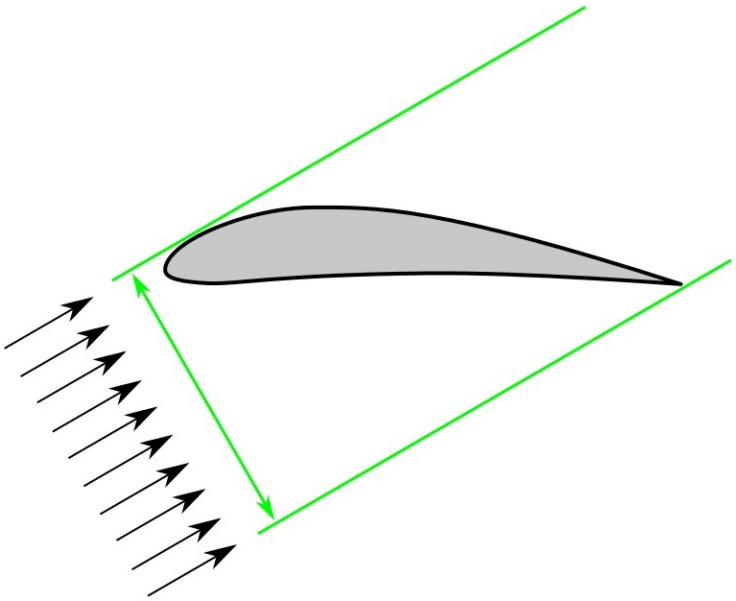


**Abbildung 3.16:** Anströmung einer Tragfläche mit einem Anstellwinkel von 30° [Koe17]

Das folgende Codebeispiel aus dem Programm (3.11) zeigt die Ermittlung der Anteile der einzelnen $C_W$-Wert-Komponenten, basierend auf der tatsächlichen Bewegung des Flugzeugs. Zu diesem Zweck wird das Verhältnis von horizontaler zu vertikaler Bewegung aus dem Bewegungsvektor auf die beiden horizontalen und vertikalen $C_W$-Werte übertragen.

```
private void CalculateActualCwAirplane()
    {
        if (movementAirplane.z > 0 && movementAirplane.z >
        movementAirplane.y)
        {
            actualCwAirplane = ((movementAirplane.y /
            movementAirplane.z * airplane.getCwAircraftForward())
               +
            ((1 - (movementAirplane.y / movementAirplane.z)) *
            airplane.getCwAircraftDown()));
        }
        else if (movementAirplane.z > 0 && movementAirplane.z <
        movementAirplane.y)
        {
            actualCwAirplane = ((movementAirplane.z /
            movementAirplane.y * airplane.getCwAircraftDown()) +
            ((1 - (movementAirplane.z / movementAirplane.y)) *
            airplane.getCwAircraftForward()));
        }
        else
        {
            actualCwAirplane = airplane.getCwAircraftForward();
        }
    }
```

## 3.12 Luftwiderstand Staudruck (Drag)

Der Luftwiderstand, auch Staudruck genannt, verhält sich sehr ähnlich zum Auftrieb (Kapitel:3.9). Die Formel für den Widerstand lautet: ([BAL11] Kapitel 1.2.2 Auftrieb und Widerstand)

$$F_A = C_W * \frac{p}{2} * v^2 * A \tag{3.4}$$

$A$ = Fläche der Tragfläche
$C_W$ = Luftwiderstandsbeiwert
$p$ = Luftdichte
$V$ = Anströmgeschwindigkeit, also die aktuelle Geschwindigkeit des Flugzeugs gesehen zur Luft.

Die Umsetzung im Programm (3.12) ähnelt stark der Auftriebsberechnung. Es wird die eben aufgeführte Formel genutzt und das Ergebnis durch einen Faktor an die Dimensionen der anderen Berechnungen angepasst. Im Falle einer nicht positiven Bewegung auf der Z-Achse wird der Auftriebswert auf Null gesetzt.

```
private void CalculateForwardDrag()
     {
     if (movementAirplane.z > 0)
     {
         currentDrag = ((actualCwAirplane* flapsFactor) *
            ((trueAirSpeed/100) * (trueAirSpeed/100)) *
            (atmosphericPressure/2));
        currentDrag *= 10;
     }
     else
     {
         currentDrag = 0;
     }
 }
```

Die Berechnung der Richtung, in welcher der Widerstand anliegt, erfolgt analog zu der in 3.10 dargestellten Funktion zur Berechnung der Auftriebsrichtung. In 3.12 wird die Richtung des Widerstands durch Bildung des Kreuzprodukts aus dem zuvor errechneten gerichteten Auftriebsvektor und einem neu erstellten zur Seite (X-Achse) zeigenden Vektor errechnet. Anschließend wird das Richtungsvektor normalisiert und mit dem Wert für den Widerstand multipliziert, um den gerichteten Widerstand zu erhalten.

```
 var dragDirectional = new Vector3(0, 0, 0);
if (movementAirplane.z != 0)
{
    dragDirection = Vector3.Cross(liftDirection, new
       Vector3(1, 0, 0)).normalized;
}
else
{
    dragDirection = new Vector3(0, 0, -1);
}
dragDirectional.x = currentDrag * dragDirection.x;
dragDirectional.y = currentDrag * dragDirection.y;
dragDirectional.z = currentDrag * dragDirection.z;
```

## 3.13 Luftdichte

Die Luftdichte beschreibt die in einem bestimmten Volumen enthaltene Masse der Luft. Angegeben wird diese durch das Symbol „$p$“. Dieser Wert nimmt mit steigender Höhe ab, steigt jedoch mit sinkenden Temperaturen leicht an. Der Temperaturunterschied wird mit steigender Höhe jedoch immer geringer. Auf Höhe des Meeresspiegels beträgt die Luftdichte bei 15°C 1,225 kgm$^{-3}$, bei 20°C sind es jedoch nur 1,2041 kgm$^{-3}$. Auf einer typischen Reiseflughöhe für größere Passagierflugzeuge mit Druckkabine von 10 km beträgt $p$ nur noch 0,412706 kgm$^{-3}$. Die Temperatur ist hier ausgehend von 15°C auf Meereshöhe auf -50°C gefallen.
Aus der Luftdichte ergibt sich der Luftdruck, welcher das Gewicht beschreibt, welches aus der Luftsäule über dem Körper entsteht. Der mittlere Luftdruck der Atmosphäre (atmosphärischer Druck) auf Meereshöhe beträgt 101 325 Pa = 101,325 kPa = 1 013,25 hPa, was einem Bar entspricht. Die Luftdichte und damit der Luftdruck nehmen in den für uns interessanten Höhen näherungsweise exponentiell ab. Dargestellt ist dies in der folgenden Grafik.

Bei 15°C auf Meereshöhe mit $p_0$=101325 Pa, was der sogenannten Standardatmosphäre entspricht, lässt sich basierend auf der barometrischen Höhenformel folgende Exponentialfunktion aufstellen: [rA17]

$$p(h) = p_0 * exp(\frac{p_0 g h}{p_0}) \tag{3.5}$$

h entspricht der Höhe, $p_0$wird auf 8000m festgelegt. Außer Acht gelassen wird hier der Temperaturunterschied bei steigender Höhe. Im Programm wird die Abnahme der Luftdichte als linear angenommen. Wie am Codebeispiel 3.17 dargestellt, wird eine stark vereinfachte Berechnung genutzt, in welcher der Luftdruck ein Faktor ist, welcher pro Höheneinheit um ein Fünftausendstel abnimmt.

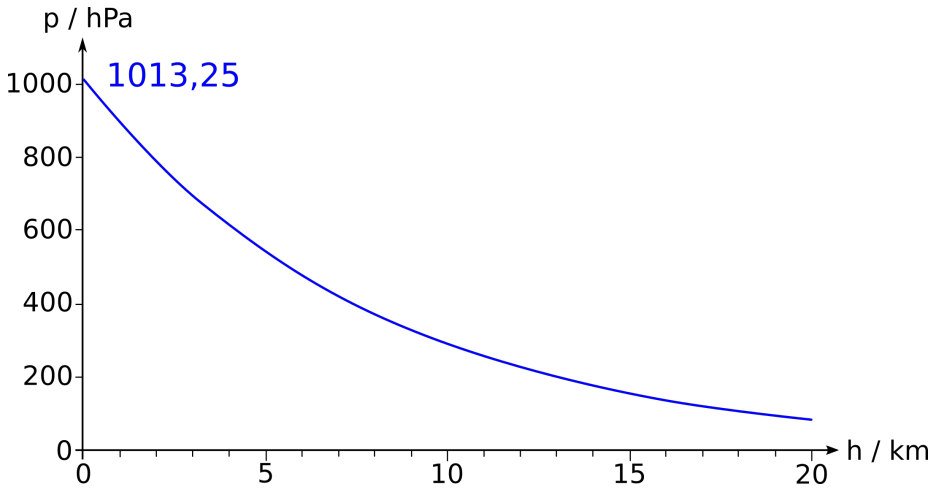



**Abbildung 3.17:** Die Abnahme des Luftdrucks mit steigender Höhe [Koe17]

```
    private void CalculateAtmosphericPressure()
{
    atmosphericPressure = 1 - (transform.position.y / 5000);
}
```

## 3.14 Schwerkraft (Gravity)

Die Schwerkraft, welche auch als Gravitation oder Massenanziehung bezeichnet wird, ist eine Kraft die zwischen zwei Körpern wirkt. Sie ist lediglich vom Gewicht der Körper abhängig, nimmt mit der Entfernung ab, lässt sich jedoch nicht abschirmen. Beide Körper ziehen sich gegenseitig an, die Beschleunigung ist umgekehrt proportional zur beschleunigten Masse. Dies bedeutet, dass die Masse von Körper A die Beschleunigung von Körper B erzeugt (und umgekehrt).
Der enorme Massenunterschied zwischen der Erde als einem Körper sowie dem Flugzeug als zweiten Körper sorgt jedoch dafür, dass die Kräfte, die vom Flugzeug auf die Erde wirken, vernachlässigt werden können. Im Programm wird daher die Gravitation als eine Beschleunigung angenommen, welche unabhängig von der Geschwindigkeit oder Ausrichtung des Flugzeugs dieses stets nach unten zur Erdoberfläche zieht. Im Programm wird der Kraftvektor von Weltkoordinaten noch auf das lokale Koordinatensystem des Flugzeugs umgerechnet, damit die Bewegung von der Ausrichtung des Flugzeugs unabhängig ist. Die Gravitationsbeschleunigung wird mit $9,81 ms^{-2}$angenommen. Die Berechnung wird in 3.14 dargestellt.

```
private void CalculateGravityV2()
{
```

```
        gravitymovement.y = -2f * 9.81f
           *airplane.getTotalWeight() * Time.deltaTime;
    }
```

Es folgt die Umwandlung des errechneten Wertes in Weltkoordinaten, damit die Schwerkraft unabhängig von der Ausrichtung des Flugzeugs wirkt.

```
        gravitymovementLocal =
           transform.InverseTransformDirection(new Vector3(0,
           gravitymovement.y, 0));
```

## 3.15 Beschleunigung und Schub (Acceleration)

Schub wird erzeugt, indem eine Stützmasse entgegen der gewünschten Bewegungsrichtung beschleunigt wird. Raketen führen diese Masse als Raketentreibstoff mit. Von luftatmenden Antrieben spricht man, wenn die Stützmasse aus der Umgebungsluft gewonnen wird. Bei beiden Varianten wird beim Verbrennen von Treibstoff chemische Energie zugeführt, welche die Stützmasse beschleunigt. Da die Leistung bei einem Kolbentriebwerk recht unabhängig von der Geschwindigkeit ist, verhält der Schub sich gleich dem Verhältnis von zugeführter Nutzleistung P zur Fluggeschwindigkeit $V_A$. Der erzeugte Schub ist also umgekehrt proportional zur Fluggeschwindigkeit.(Siehe Gleichung 3.7) ([BAL11] Kapitel 1.2.5 Antrieb)
Die Formel für den Schub eines Kolbentriebwerks mit $\eta$ als Propellerwirkungsgrad lautet:

$$F = \eta * \frac{P}{V_A} \tag{3.6}$$

Diese Formel wird im Programm zur Schubberechnung genutzt, unabhängig davon, ob der Propeller (Prop) oder die Turbine (Fan) gewählt wird. Lediglich die Leistung unterscheidet sich. Falls die aktuelle Fluggeschwindigkeit nicht größer als null ist, fällt sie aus der Berechnung heraus, da sonst durch null geteilt werden müsste.

```
private void CalculateThrust()
        {
        if (trueAirSpeed > 0)
        {
            totalThrust = (currentEnginePower / trueAirSpeed/2) *
               atmosphericPressure;
        }
        else
        {
            totalThrust = currentEnginePower *
               atmosphericPressure;
        }
    }
```

Diese Berechnung ist sehr ungenau, leider ist es mir nicht gelungen auf Basis des Eulerverfahrens oder Runge Kutter eine funktionierende Geschwindigkeitsberechnung zu implementieren. Das Ergebnis war stets ein nicht nutzbares instabiles Verhalten, welches in der Sektion Schwierigkeiten und Probleme (3.18) beschrieben wird.
Nachfolgend die Formel zur Errechnung des Schubbelastungsgrads:

$$C_{Fp} = \frac{F}{P_0 * S_F} \tag{3.7}$$

$p0$ = Aussendruck
$S_F$ = Triebwerksstirnfläche
$C_{Fp}$ = statischer Schubbelastungsgrad

Nachfolgend eine Grafik in Anlehnung an das Buch Flugregelung mit verschiedenen Antriebssystemen. In dem Prototypen werden der Prop (Propeller mit Kolbenantrieb), sowie der Fan (Düsenantrieb) betrachtet, bei beiden wird von keiner Veränderung über den Geschwindigkeitsbereich ausgegangen, in der Grafik entspräche dies einer waagerechten Linie.

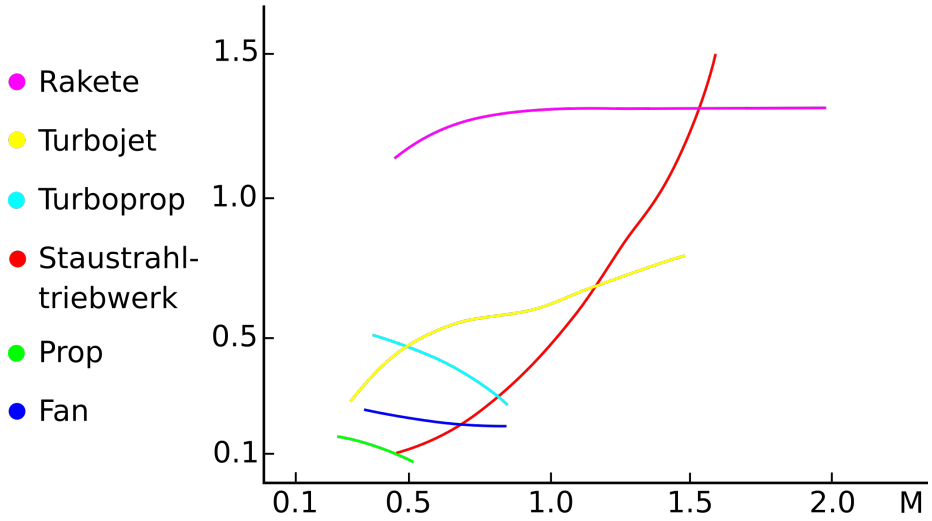



**Abbildung 3.18:** Der erzeugte Schub verschiedener Antriebssysteme im Verhältnis zur Geschwindigkeit (Machzahl: M) [BAL11]

## 3.16 Schwerpunkt und Auftriebspunkt (Center of Mass and Lift)

Aus den Gewichtswerten der einzelnen Komponenten ergibt sich das Gesamtgewicht des Flugzeugs. (Siehe Kapitel: 3.5) Der Angriffspunkt des kombinierten Gewichtsvektors wird als Schwerpunkt bezeichnet. Seine Position ergibt sich aus den Einzelgewichten, sowie deren Abstand zur Mitte. Ein weiter von der Mitte entfernt liegendes Gewicht beeinflusst die Position des Schwerpunkts stärker als ein gleichgroßes Gewicht, das zentraler liegt. Der Schwerpunkt hängt demnach von der Auswahl der einzelnen Komponenten ab, allerdings ändert sich seine Position auch durch die Beladung des Flugzeugs (Fracht, Passagiere und Treibstoff) Ein Flugzeug ist in der Regel symmetrisch um die Längsachse(Z-Achse) aufgebaut, Beladung und Betankung sowie der Verbrauch des Treibstoffs erfolgen nach Möglichkeit auch gleichmäßig, somit liegt der Schwerpunkt zentriert um die Querachse(X-Achse) und ist lediglich um die Hoch(Y-Achse)- und Längsachse(Z-Achse) variabel ([BAL11] 1.2.3 Momentengleichgewicht).
In dem Programm wird die Position des Schwerpunkts jedoch nicht errechnet, er liegt unveränderlich auf dem Pivotpunkt des Modells, es werden lediglich die Einzelgewichte addiert.
Der Auftriebspunkt ergibt sich zum größten Teil aus der Position sowie der Form der Tragflächen. Allerdings wird er im Flug durch den jeweiligen Flugzustand beeinflusst. So erzeugt das neutral geformte Höhenruder bei erhöhtem Anstellwinkel zusätzlichen Auftrieb, der den Gesamtauftrieb erhöht und somit den Auftriebspunkt etwas zum Flugzeugheck wandern lässt. Im Gegensatz zum Schwerpunkt ist der Auftriebspunkt in der Querachse(X-Achse) nicht immer zentriert. Im Kurvenflug erzeugen die Tragflächen unterschiedlich viel Auftrieb, was den Auftriebspunkt entlang der Querachse(X-Achse)in Richtung der weniger waagerechter stehenden Tragfläche verschiebt. (Siehe Kapitel: 3.9)
Auch der Auftriebspunkt wird in dem Programm nicht errechnet, er befindet sich genau wie der Schwerpunkt unveränderlich am Pivotpunkt des Flugzeugs.
Der Schwerpunkt liegt im stationären Geradeausflug leicht vor dem Auftriebspunkt, bei steigendem Anstellwinkel wandert der Auftriebspunkt jedoch nach vorne, das Momentgleichgewicht muss jetzt durch das Höhenruder ausgeglichen werden.

## 3.17 Landschaft und Wolken

Da der Fokus in dieser Arbeit auf der Flugphysik und weniger auf der optischen Darstellung der Landschaft oder der Modelle liegt, besteht die Landschaft aus einzelnen Komponenten, die größten Teils bereits in den Demoprojekten, die der Unityengine beiliegen, vorhanden sind. Ein weiterer wichtiger Faktor ist der starke Einbruch der Performance, wenn eine zu detailreiche Umgebung dargestellt wird.
Die Szene besteht aus einer inselförmigen Landschaft. Die Größe des Terrains wurde

stark hoch skaliert, um dem Maßstab des Flugzeugs und der davon abhängigen Bewegung zu entsprechen. Automatisch auf dem Terrain generierte Bäume stellen die einzige Vegetation dar. Umgeben ist die Insel von Wasserkacheln, welche ebenfalls aus den Beispielszenen stammen. Die genutzte Skybox erzeugt die Illusion einer sehr viel detailreicheren Umgebung, sie ist mit der Sonnenlichtquelle ausgerichtet, was zur Position der dargestellten Sonne passende Beleuchtung sowie Schattenwürfe ermöglicht.
Etwas aufwändiger ist die Erstellung der zusätzlichen Wolken sowie des Nebeleffekts. Diese bestehen aus Partikelemittern, die in vorgegebenen Intervallen die Wolkentextur erzeugen. Durch Variation der Ausrichtung sowie einer Änderung der Größe über die begrenzte Lebenszeit dieser Partikel entsteht der Eindruck von volumetrischem Rauch oder Wolken, welche sich auch in sich bewegen.
Im Gegensatz zu den statischen Wolken der Skybox lassen sich die selbst erstellen

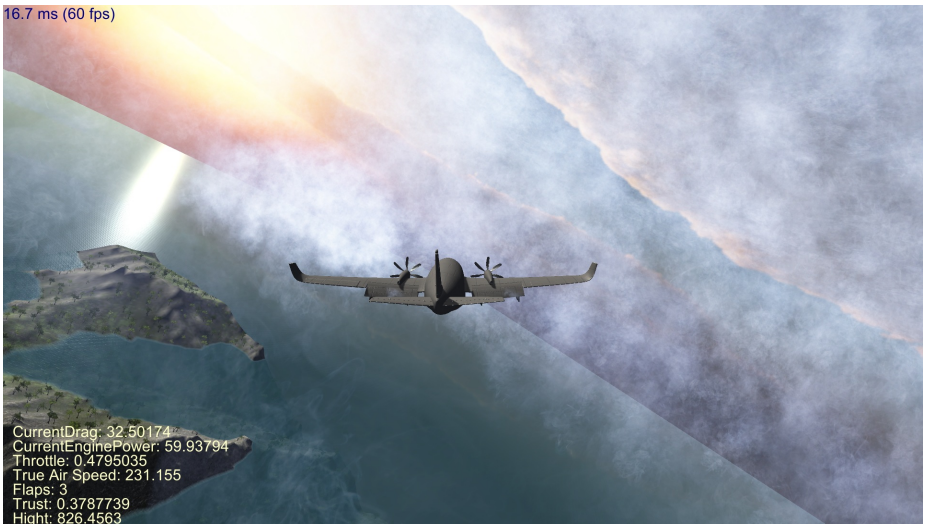
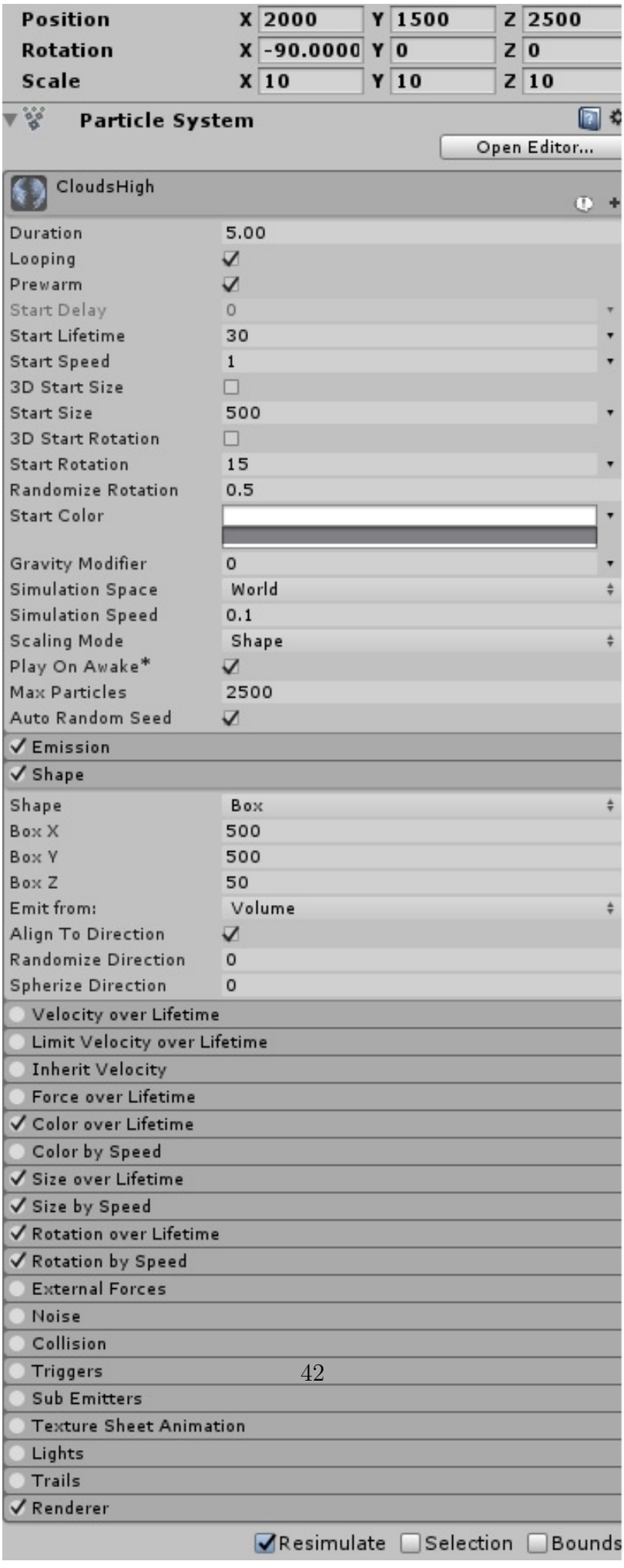



**Abbildung 3.19:** Flug durch eine Wolke [Koe17]

Wolken durchfliegen (Abbildung 3.19), da sie eine von der Emittergröße und Position abhängige Form und Größe haben. Sie vermitteln so bei größerer Höhe immer noch ein Gefühl für die aktuelle Geschwindigkeit. In der nachfolgenden Grafik 3.20) wird ein Teil der Einstellmöglichkeiten dargestellt, hier am Beispiel der hohen Wolkenschicht, welche einen Großteil der Insel überragt. Auf sehr ähnliche Art wird auch den leichten Abgaseffekt bei den Triebwerken erstellt, auch hier werden Rauchpartikel erzeugt, die sich über Zeit ausdehnen und durch einen steigenden Alphawert durchsichtiger werden bis sie ganz verschwinden. Durch die Bewegung des Flugzeugs entsteht so je nach Einstellung ein leichter Kondenzstreifen oder nur eine kleine Abgaswolke.


Position X 2000 Y 1500 Z 2500
Rotation X -90.0000 Y 0 Z 0
Scale X 10 Y 10 Z 10
Particle System
Open Editor...
CloudsHigh
Duration 5.00
Looping
Prewarm
Start Delay 0
Start Lifetime 30
Start Speed 1
3D Start Size
Start Size 500
3D Start Rotation
Start Rotation 15
Randomize Rotation 0.5
Start Color
Gravity Modifier 0
Simulation Space World
Simulation Speed 0.1
Scaling Mode Shape
Play On Awake*
Max Particles 2500
Auto Random Seed
Emission
Shape
Shape Box
Box X 500
Box Y 500
Box Z 50
Emit from: Volume
Align To Direction
Randomize Direction 0
Spherize Direction 0
Velocity over Lifetime
Limit Velocity over Lifetime
Inherit Velocity
Force over Lifetime
Color over Lifetime
Color by Speed
Size over Lifetime
Size by Speed
Rotation over Lifetime
Rotation by Speed
External Forces
Noise
Collision
Triggers
Sub Emitters
Texture Sheet Animation
Lights
Trails
Renderer
Resimulate
Selection
Bounds

## 3.18 Schwierigkeiten und Probleme

Eine immer wieder auftretende Schwierigkeit bei der Erstellung des Programms war die optisch passende Abstimmung der einzelnen Flugzeugeigenschaften untereinander. Zu diesen zählen das Gewicht der einzelnen Komponenten, ihre $C_W$-Werte und Flächen, der Auftrieb der Tragflächen sowie der Schub der Triebwerke. Da das Programm ohne Einheiten arbeitet, sind die genutzten Werte frei erfunden und ihre Dimensionen durch Ausprobieren entstanden. Die Änderung eines Wertes wirkt sich ebenfalls auf die anderen Werte aus, ebenso wie die Änderung einer Berechnung, die auf einen oder mehrere Werte zugreift. Dies erfordert eine ständige Überprüfung und gegebenenfalls Korrektur nach jeder Veränderung eines Werts. Auch der Test neu implementierter Funktionen wird so erschwert, da bei ungewolltem Verhalten nicht zwingend die Funktion fehlerhaft sein muss, ein unerwünschtes Ergebnis kann auch an falsch dimensionierten Werten liegen. Hier würde das Nutzen von Standardeinheiten Abhilfe schaffen. Dies würde es ermöglichen, reale Werte mit SI Einheiten zu nutzen, was diese als Fehlerquelle ausschließen würde.
Ein Problem, über dessen Ursache ich nur spekulieren kann, ist ein gelegentlich auftretendes extremes Zittern. Das Flugzeug scheint sich sehr stark aufzuschaukeln, bis die Werte den Wertebereich verlassen und das Programm sich beendet. Der Effekt tritt bei aktuellen Einstellungen leicht beim Start auf, fängt sich dann allerdings nach kurzer Zeit. Meine Vermutung ist, dass sich die gegenseitig beeinflussenden Werte im ungünstigen Fall nicht stabilisieren, sondern aufschaukeln. So steigt durch die Beschleunigung die Geschwindigkeit, was den Luftwiderstand erhöht, wodurch die Geschwindigkeit wieder reduziert wird. Von der Geschwindigkeit ist auch direkt der Auftrieb abhängig, wodurch das Flugzeug dann nicht nur in horizontaler Richtung, sondern auch vertikal zu springen beginnt.

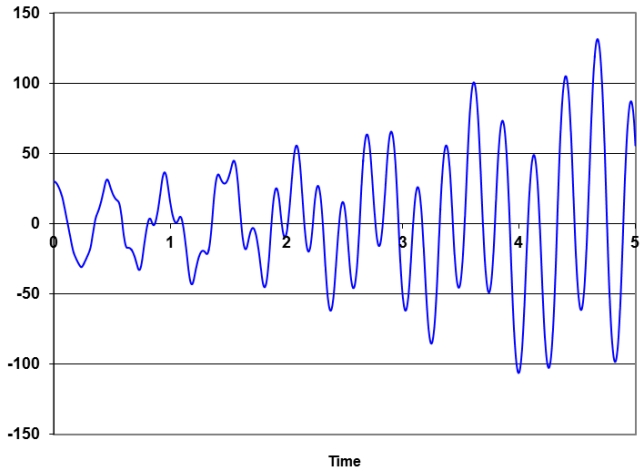



**Abbildung 3.21:** Beispiel eines instabilen Eulerverfahrens [BH13]

Die Recherche zu dem Thema ([BH13] Kapitel: Real Time Simulations) ergab, dass das zur Geschwindigkeitsberechnung häufig genutzte Eulerverfahren instabile Ergebnisse liefern kann. Ich setze es zwar nicht bewusst ein, allerdings entsprechen die Beschreibungen sehr genau meinen Beobachtungen. Abbildung 3.21 zeig die grafische Darstellung, eines instabilen Eulerverfahrens.

Zur Milderung des Problems habe ich eine Funktion geschrieben, welche die letzten 30 errechneten Bewegungswerte in einem Array speichert und den Mittelwert ausgibt, so wird leider nicht die Ursache des Problems beseitigt, aber die Folgen werden stark abgemildert.
Nachfolgend die Funktion:

```
    private void SmoothMoving()
{
    if (frameCounter <= (moveArray.Length-2))
    {
        frameCounter++;
    }
    else
    {
        frameCounter = 0;
    }

    moveArray[frameCounter] = movementAirplane;

    int tempCounter = 0;
    for (int i = 0; i < moveArray.Length; i++)
    {
        if (moveArray[i] == new Vector3(0, 0, 0)
         || moveArray[i] == null)
        {
            //keine Veränderung / leerer Wert
        }
        else
        {
            tempCounter++;
            movementAirplane += moveArray[i];
        }
    }
    movementAirplane = (movementAirplane / tempCounter);
}
```

Ebenfalls unangenehm fällt ein häufiges Springen der Framerate auf. Durch den Einsatz eines multiplikativen Zeitfaktors sollen unterschiedlich lange Berechnungszeiten der einzelnen Frames ausgeglichen werden. Dieser Faktor ist die Zeit, die der letzte Frame zur Berechnung benötigt hat. Sämtliche Berechnungen und Bewegungen werden deshalb mit dem Faktor multipliziert.

Eine weitere große Schwierigkeit ist die Berechnung von Kollision, welche eine Voraussetzung zur Umsetzung von Start und Landungen sowie Absturzerkennung ist. Die Unity Engine bietet ein bereits implementiertes Physiksystem. Hierzu müssen von der Physik beeinflusste Objekte eine Rigidbody Komponente erhalten. Diese errechnet ohne weiteres Zutun Schwerkraft und, falls die passenden Collider Komponenten den Objekten hinzugefügt wurden, auch Kollisionen. Diese intern ablaufenden und nicht direkt einsehbaren Berechnungen erzeugen aber in Kombination mit den eigenen selbst erstellten Physikberechnungen ein ungewolltes Verhalten. Für dieses Problem gibt es zwei mögliche aber jeweils sehr zeitaufwändige Lösungen. Die erste Variante wäre es, die gesamte Physikberechnung auf die Nutzung von Rigidbody umzustellen. Dies widerspricht allerdings der Zielsetzung dieser Arbeit, die gesamten Berechnungen selbst durchzuführen und keine fertige Blackbox zu nutzen. Auch der Zeitaufwand für eine Umstellung wäre sehr hoch, da jede Berechnung einzeln angepasst werden müsste.
Die zweite Variante wäre eine selbst erstellte Kollisionsberechnung. Auch diese Möglichkeit wäre voraussichtlich sehr zeitaufwändig. Zügig umsetzbar wäre höchstens eine reine Erkennung vom Kontakt zwischen Flugzeugmodell und Landschaft, allerdings ohne weitere physikalische Auswirkungen.

## 3.19 Mögliche Erweiterungen

Die Möglichkeiten zur Erweiterung des Programms sind vielfältig. Als wichtig erachte ich eine Lösung der im Kapitel Probleme und Lösungen (3.18) aufgeführten Punkte: Diese sind die Umstellung aller Berechnungen auf SI Einheiten zur leichteren Erweiterbarkeit des Programms sowie einfacheren Vergleichbarkeit mit reellen Größen tatsächlich existierender Flugzeuge. Ein weiterer wichtiger Punkt ist eine realistischere Geschwindigkeitsberechnung, beispielsweise mit dem erweiterten Eulerverfahren oder Runge Kutter. (Der bisherige Versuch, dies zu implementieren, ist leider gescheitert). Das dritte große Problem, welches behoben werden sollte, ist die Kollisionserkennung, welche eine Voraussetzung zur Implementierung von Starts und Landungen auf Flughäfen sowie einer Absturzerkennung ist.
Ein großer Punkt ist die Verbesserung der Simulation von Kräften und allgemein der Berechnungen von daraus resultierenden Effekten. Die bisherige Berechnung geht von einem einzigen Punkt aus, an dem alle Kräfte zusammengefasst wirken. Durch die Aufteilung der Berechnung der Tragflächeneigenschaften in mindestens die einzelnen Tragflächen, besser aber noch einzelne Tragflächensektionen, ergeben sich neue Effekte, welche die bisherigen Annäherungen ersetzen. (Siehe Kapitel Tragflächensegmentierung 3.6, sowie Tragflächenausrichtung 3.7). Daraus folgt die Möglichkeit Schwerpunkt und Auftriebspunkt zu errechnen, sowie deren Verhalten in unterschiedlichen Fluglagen zu simulieren. (Kapitel 3.16)
Eine Simulation verschiedener Wetterlagen, vor allem von Wind verschiedener Stärken und Richtungen, abhängig von der jeweiligen Flughöhe sowie der Topographie der Landschaft stellen eine Sinnvolle Erweiterung dar. Weitere Erweiterungen mit geringerer Priorität sind eine umfassendere Implementierung von weiteren Soundeffekten sowie eine Erweiterung und genauere Ausarbeitung der Landschaft. Auch weitere detailliertere Flugzeugkomponenten sind möglich.

# 4 Alternativen und Bestehendes

## 4.1 Geschichte der Flugsimulatoren

Die ersten Flugsimulatoren beschränkten sich auf eine Simulation des Instrumentenflugs (Es wird nur nach Instrumenten geflogen, der Pilot sieht also lediglich die Anzeigen im Cockpit und hat keinen Blich nach draußen), da die zur Verfügung stehende Grafikleistung für Privatanwender nicht für eine graphische Darstellung der Umgebung reichte. Als der erste grafische Flugsimulator überhaupt gilt das 1980 von der Firma subLOGIC veröffentlichte Programm „Flight Simulator“, welches für den Apple II entwickelt wurde und erstmals Sichtflug (Der Pilot kann sich an der Umgebung orientieren) ermöglichte. Sein Nachfolger „Flight Simulator II“ wurde 1982 an Microsoft lizensiert und erschien somit für den IBM-PC und 1984 für den Commodore 64 und Atari XL. Auf dieser Grundlage entwickelte Microsoft den „Microsoft Flight Simulator“ (Siehe Kapitel: 4.3.2)


**Abbildung 4.1:** Ein Landeanflug im Flight Smulator II [sub82]

Während die Flugphysik bereits größtenteils erforscht war, steckte die Computergrafik und Rechenleistung noch in der ersten Anfängen. Dies führte dazu, dass das Flugverhalten eines Flugzeugs, in diesem Fall einer einmotorigen Piper PA-28-181

Archer II, sehr genau berechnet werden konnte, die optische Darstellung im Raum war allerdings noch sehr rudimentär.
Der „Flight Simulator II“ bot ein nahezu vollständig ausgestattetes und funktionsfähiges Cockpit. Somit war ein reiner Instumentenflug (IFR) möglich, sogar Funkfeuer konnten angepeilt werden. Die 10.000 mal 10.000 Meilen große Landschaft besteht aus einer flachen Ebene, auf der Berge durch Polygonobjekte dargestellt werden. Gewässer sowie vereinzelte Gebäude werden ebenso wie natürlich Flughäfen dargestellt. Im Flug wird der Bildschirm zweigeteilt dargestellt, im oberen Bereich befindet sich der Blick aus dem Cockpit, welcher mit vier Farben im Multicolor Modus dargestellt wird, darunter befindet sich das Cockpit mit allen für den Flug relevanten Anzeigen. Dieses nutzt den HiRes Modus, welcher eine höhere Auflösung bietet, jedoch nur 2 Farben pro Kachel darstellen kann.
Es werden zwei Wolkenschichten simuliert, ebenso wie bis zu vier unterschiedliche Windebenen. Es gibt drei Tageszeitmodi: Tag, Nacht und Dämmerung. Die Bildrate beträgt ungefähr ein Bild pro Sekunde, was für aktuelle Verhältnisse ein sehr stockendes Gefühl vermittelt. Aktuell werden in der Regel 30 Bilder pro Sekunde angestrebt, bei schnellen Bewegungen haben 60 Bilder oder mehr jedoch deutliche Vorteile, da die Darstellung als flüssiger wahrgenommen wird. 1990 wurde „Red Baron“ von Dynamix


**Abbildung 4.2:** Das Spiel Red Baron [Dyn90]

als DOS Spiel veröffentlicht, es ist der erste Flugsimulator, welcher mehrere Spieler darstellen kann und deren Flugzeuge als dreidimensionale Vektorgrafiken rendert. Die Auflösung beträgt genau wie beim „Flight Simulator II“ 320 mal 200 Pixel, allerdings nutzt „Red Baron“ eine 256 Farben Darstellung. Der Spieler fliegt hier Flugzeuge aus der Zeit des ersten Weltkriegs. Die Möglichkeiten der Simulation waren schon sehr hoch, so konnte der Pilot bei zu hohen Belastungen das Bewusstsein verlieren oder

der Vergaser des Motors einfrieren.
Ein großer Sprung im Realitätsgrad der Simulationen war 1995 das Erscheinen von „Flight Unlimited“ von Looking Glass. Hier wurde erstmals numerische Strömungsmechanik (englisch: Computational Fluid Dynamics, CFD) eingesetzt. Somit ist eine genaue Simulation der Eigenschaften des Flugzeugs bei unterschiedlichen und extremen Flugzuständen möglich. (Siehe Kapitel Anströmrichtung: 3.10 sowie Kapitel Tragflächenausrichtung: 3.7) Der Spieler fliegt hier ein Kunstflugzeug, was von der genaueren Simulation stark profitiert. Die beiden Nachfolger nutzen die Technik nicht mehr, jedoch war bei ihnen die optische Darstellung deutlich verbessert, da hier erstmals kein riesiges Areal mit stark begrenztem Detailgrad dargestellt wurde, sondern sich auf einen kleinen Bereich beschränkt wurde, der dafür jedoch auf der Grundlage von Satellitenaufnahmen entstanden ist. Diese Technik wird bei aktuellen Simulationen häufig für erweiterte Szenerien genutzt, die in der Regel durch ihre gewaltige Datenmenge jedoch erst später einzeln ausgewählt und hinzugeladen werden.

**Abbildung 4.3:** Flight Unlimited 3 [Stu99]

## 4.2 Baukästen und Arcadespiele

### 4.2.1 Simple Planes

Simple Planes ist ein kleines Baukastenspiel, mit dem sich Fahrzeuge (vor allem Flugzeuge) aus sehr einfachen aber relativ frei anpassbaren Formen und Komponenten zusammen stellen lässt. Es bietet große Freiheiten in der Gestaltung der Fahrzeuge. Da die Eigenschaften aber relativ realistisch berechnet werden und sich je nach Position und Form ändern, ist jedoch für eine funktionsfähiges Fahrzeug eine durchdachte Konstruktion nötig.
Simple Planes greift die Grundidee hinter meinem Programm sehr gut auf, legt den Fokus allerdings noch stärker auf Individualisierungsmöglichkeiten einzelner Komponenten, als es mir möglich war.


**Abbildung 4.4:** Das Simple Planes Logo [Jun17]

### 4.2.2 Kerbal Space Program

Das von Squad entwickelte Spiel Kerbal Space Program nutzt die Unity Engine. Es lässt den Nutzer aus unzählig vielen verschiedenen Bauteilen ein Flugzeug oder eine Rakete erstellen. Die Positionierung der Komponenten ist sehr frei und ihre Auswirkungen sind genau simuliert. Ziel des Spielers ist es, das Weltall zu erreichen, es lässt sich aber sogar auf dem Mond landen. Der Fokus liegt auf der Erstellung funktionierender Fahrzeuge. Das Spiel ist sehr gut durch Mods zu erweitern, was zu etlichen von der Community erstellten Modifikationen geführt hat, die den Funktionsumfang und die Komplexität erheblich erweitern.


**Abbildung 4.5:** Das Kerbal Space Program Logo [Squ17]

### 4.2.3 Besige


**Abbildung 4.6:** Das Besiege Logo [Spi17]

Besiege ist ein Baukastenspiel des Entwicklerstudios Spiderlinggames, in dem der Spieler sich eigene Fahrzeuge und Objekte aus fertigen Blöcken erstellen kann. Die Bauteile haben alle physikalische Eigenschaften, welche das Verhalten des Objekts

beeinflussen. Es gibt einfache Strukturblöcke und Streben, aber auch Funktionsteile wie Räder, Kolben oder Flügel. Neben der Auswahl der Bauteile hat auch ihre Position Einfluss auf die Berechnungen.

## 4.3 Flugsimulatoren

### 4.3.1 X-Plane


**Abbildung 4.7:** Ein bei X-Plane 11 verwendetes Logo [Res17]

X-Plane ist ein sehr auf Realismus ausgelegter Flugsimulator von Laminar Research. Aktuell (Frühjahr 2017) befindet Version X-Plane 11 in einem Betatest.
Der Anspruch eine realistische Simulation zu erschaffen spiegelt sich sowohl in der Zulassung zum realen Pilotentraining als auch in einer recht hohen Einstiegshürde für Laien wieder. Der Simulator wird von einer Reihe von Entwicklern unterstützt, die Erweiterungen und Flugzeuge mit zum Teil enormem Detailgrad anbieten.

### 4.3.2 MS Flightsimulator


**Abbildung 4.8:** Das Logo des FSX in der Steamversion [Mic17]

Der Microsoft Flightsimulator ist ist einer der bekanntesten Flugsimulatoren überhaupt. Seine Vorläufer gehen auf die Anfänge der Flugsimulatoren für Heimrechner zurück (Siehe Kapitel: Geschichte der Flugsimulatoren 4.1). Er ist zuletzt 2006 in der Version Flightsimulator X erschienen. Dieser wurde ohne inhaltliche Änderungen lediglich als Steamversion neu aufgelegt. Weitere Versionen sind nach bisherigen Ankündigungen nicht geplant. Der Rüstungskonzern Lockheed Martin hat die vollständige Lizenz aufgekauft und entwickelt seit dem unter dem Namen Prepar3D eine eigenständige Simulation, die sich allerdings explizit nicht als Spiel versteht und sogar in einer Version für militärisches Training verfügbar ist.
Der Flightsimulator X verfügt über recht realistische Einstellungen, sein Fokus liegt allerdings auf der Unterhaltung. Es lassen sich unzählige Erweiterungen finden, die den Simulationscharakter teils deutlich stärken.

# 5 Fazit

Die Beantwortung der eingangs gestellten Frage, ob sich mit einfachen Mitteln und beschränkten Vorkenntnissen ein realistisch wirkender Flugsimulator erstellen lässt, hängt von der Auslegung des Wortes „realistisch“ ab. Der Aufwand und die nötigen Kenntnisse, um eine Simulation zu erstellen, welche die Eigenschaften eines Flugzeugs physikalisch korrekt simuliert, ist enorm und übersteigt meine Möglichkeiten bei Weitem. Zur Bewertung des Ergebnisses muss daher der Kompromiss zwischen Aufwand und optischer Wirkung betrachtet werden.
Durch die zum Teil sehr starke Vereinfachung der Berechnungen kann der Begriff Simulation nicht mehr genutzt werden. Trotzdem ist es mir gelungen, viele wichtige Zusammenhänge zwischen den Flugzeugeigenschaften und dem Flugverhalten zu implementieren: Die durch den Schub der Triebwerke erzeugte Fluggeschwindigkeit beeinflusst den Auftrieb sowie den Luftwiderstand. Dieser hängt neben der Geschwindigkeit jedoch auch vom Anströmwinkel der Luft ab. Das Gewicht beeinflusst die Schwerkraft, welche durch den Auftrieb ausgeglichen werden muss. Somit haben die Eigenschaften Motorleistung(Schub), Luftwiderstand ($C_W$-Wert und Fläche) und Gewicht der ausgewählten Flugzeugkomponenten direkten Einfluss auf das Flugverhalten des Flugzeugs. Ein Flugzeug mit erhöhtem Gesamtgewicht benötigt zum Beispiel mehr Fluggeschwindigkeit um genügend Auftrieb zu erzeugen. Ein geringerer Luftwiderstand verringert den benötigten Schub für die selbe Geschwindigkeit und Tragflächen mit höherem Auftrieb erzeugen auch mehr Luftwiderstand. Ich komme somit zu dem Ergebnis:
**Es ist möglich als einzelne Person ohne fundierte physikalische Vorkenntnisse und mit nur mäßiger Programmiererfahrung innerhalb einer Bachelorarbeit ein recht realistisch wirkendes Flugverhalten darzustellen.**
Dieses stellt allerdings keine korrekte Simulation dar, kann aber wie im Kapitel mögliche Erweiterungen: (3.19) beschrieben stark ausgebaut und erweitert werden, was den Simulationscharakter verstärken würde. Der Arbeitsaufwand variiert dabei jeweils stark, abhängig von den jeweiligen Vorkenntnissen in den drei Bereichen allgemeine Programmierung, Kenntnisse der Unity Engine sowie Kenntnisse im Bereich Flugphysik. In meinem Fall hat die Recherche von Funktionen in der Unity Dokumentation viel Zeit in Anspruch genommen. Die grundlegenden Zusammenhänge der Flugphysik waren mir jedoch bereits durch eine frühere Segelflugausbildung bekannt, weshalb eine genaue Recherche in dem Bereich erst nach der grundlegenden Erstellung des Programms und in Vorbereitung zu dieser schriftlichen Ausarbeitung erfolgte. Dieser Punkt birgt deutliches Verbesserungspotenzial im Arbeitsablauf. Das durch die Recherchen vertiefte Verständnis der genauen Zusammenhänge hätte ei-

ne schnellere Umsetzung im Programm ermöglicht, auch wenn glücklicher Weise die grundsätzlichen Zusammenhänge zuvor bereits korrekt umgesetzt wurden.
Die Auswahl der Unity Engine als Plattform bietet einem Vorteile durch eine umfassende Dokumentation sowie eine große Community mit vielen frei verfügbaren Beispielen und Anleitungen. Die Engine an sich ist leicht zu bedienen und ermöglicht eine intuitive Arbeitsweise. Ein negativer Punkt sind zufällig auftretende Kompilierungsfehler, welche nicht reproduzierbar waren, nach einem Neustart der Enginge jedoch verschwunden sind. Ärgerlich ist dies besonders in Kombination mit bei der Erstellung neuer Funktion tatsächlich vorhandenen Fehlern. Praktisch ist die Fähigkeit von Unity, in Cinema 4D erstellte Modelle vom typischen .c4d- Format in das benötigte .fbx- Format beim Importieren zu konvertieren. Die Studentenversion von Cinema 4D erwies sich als gutes Werkzeug zur Erstellung der Flugzeugkomponenten, hier war jedoch meine bisherige Erfahrung in der Nutzung ausschlaggebend, andere Programme wie Blender oder 3ds Max sollten gute Alternativen darstellen.

# Abbildungsverzeichnis

# Literaturverzeichnis


[BAL11] Rudolf Brockhaus, Wolfgang Alles, and Robert Luckner. *Flugregelung*. Springer-Verlag GmbH, 2011.

[BH13] David M. Bourg and Kenneth Humphreys. *Physics for Game Developers*. O'Reilly Media, Inc, USA, 2013.

[Dyn90] Dynamics. Red baron, 1990. Zugriff am 06.06.2017 www.mobygames.com.

[Jun17] Jundroo. Simple planes logo, 2017. Zugriff am 06.06.2017 https://www.simpleplanes.com/Content/presskit/SimplePlanes-Icon.png.

[Kas03] Winfried Kassera. *Flug ohne Motor. Ein Lehrbuch fuer den Segelflieger*. Motorbuch Verlag Pietsch, 2003.

[Koe17] Stephan Koehler. Eigene grafiken, 2017. Selbst erstellte Grafik oder Screenshot.

[Max17] Maxon. Cinema 4d studentenversion, 2017. Zugriff am 06.06.2017 https://www.maxon.net/de/training/studentenversion/.

[Mic17] Microsoft. Microsoft flight simulator logo, 2017. Zugriff am 06.06.2017 http://cdn.akamai.steamstatic.com/steam/apps/314160/header.jpg?t=1496391309.

[Ost12] Wolfgang W. Osterhage. *Studium Generale Physik*. Spektrum Akademischer Verlag, 2012.

[rA17] rwth Aachen. Barometrische hoehenformel, 2017. Zugriff am 06.06.2017 https://web.physik.rwth-aachen.de/ fluegge/Vorlesung/PhysIpub/Exscript/8Kapitel/IIX6Kapitel.html.

[Res17] Laminar Research. X-plane-logo, 2017. Zugriff am 06.06.2017 http://www.x-plane.com/wp-content/uploads/2016/10/x-plane-11-announcement-video.jpg.

[Spi17] Spiderlinggames. Besige logo, 2017. Zugriff am 06.06.2017 http://spiderlinggames.co.uk/BesiegeTitle.png.

[Squ17] Squad. Kerbal space program logo, 2017. Zugriff am 06.06.2017 ksp.multiplay.com.

[Stu99] Looking Glass Studios. Flight unlimited 3, 1999. Zugriff am 06.06.2017 http://www.flightforum.ch/board/index.php?/topic/32688-die-flight-unlimited-serie-von-looking-glass-studios/page-2.

[sub82] subLOGIC. Flight simulator 2, 1982. Zugriff am 06.06.2017.

[Tec17a] Unity Technologies. Scriptreference, 2017. Zugriff am 06.06.2017 https://docs.unity3d.com/ScriptReference/.

[Tec17b] Unity Technologies. Unity download, 2017. Zugriff am 06.06.2017 https://unity3d.com/de/get-unity/download.

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum Stephan Köhler